OSKILLOSKOOPPI

Roope Kokkoniemi

Opinnäytetyö
Toukokuu 2012
Tietotekniikka
Sulautetut järjestelmät

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Sulautetut järjestelmät

ROOPE KOKKONIEMI:
Oskilloskooppi

Opinnäytetyö 55 sivua, josta liitteitä 27 sivua
Toukokuu 2012

Työn tavoitteena oli rakentaa harrastelijan tarpeisiin riittävä, mutta kuitenkin kohtalaiseen mittausnopeuteen kykenevä oskilloskooppi. Työn suurimpana haasteena oli USB-yhteyden muodostaminen laitteen ja tietokoneen välille, sekä mitattavan signaalin muokkaaminen siten, että se voitiin mitata mikrokontrollerin AD-muuntimella.

Työssä suunniteltiin ja rakennettiin tulovahvistin, jonka avulla voidaan käyttää tavallista oskilloskoopin mittapäätä signaalin mittaamiseen. Mikrokontrollerille suunniteltiin ohjelmisto, jolla voidaan lukea AD-muuntimen tuloksia ja siirtää ne USB-yhteydellä tietokoneelle.

Tietokoneelle suunniteltiin ohjelmisto, jolla voidaan vastaanottaa muunnostuloksia mikrokontrollerilta. Ohjelmistoon suunniteltiin oskilloskoopin lisäksi spektrianalysaattori. Lisäksi ohjelmistoon suunniteltiin yksinkertaisia mittausominaisuuksia. Toteutukseen käytettiin Qt-kehitysympäristöä ja C++-ohjelmointikieltä.

Asiasanat: oskilloskooppi, spektrianalysaattori, dso.

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technology
Embedded Systems

ROOPE KOKKONIEMI:
Oscilloscope

Bachelor's thesis 55 pages, appendices 27 pages
May 2012

Purpose of this thesis is to build an oscilloscope with minimal costs but at same time adequate for hobbyist. Major challenge of this thesis was to establish USB connection between microcontroller and computer. Also the signal to be measured had to be modified before it could be measured by the microcontroller.

In this thesis, a preamplifier is build, with which regular oscilloscope probe can be used to measure signal. Software is designed for microcontroller that can read AD converter results and transmit them to computer over USB connection.

Software is designed for computer that can receive AD converter results from microcontroller. Oscilloscope and spectrum analyzer is designed for software. Also some simple measuring functionality is designed. Qt development environment and C++ programming language is used to implement software.

Key words: oscilloscope, spectrum analyzer, dso.

SISÄLLYS

1 JOHDANTO

Oskilloskooppi on elektroniikan kehityksessä ja vianetsinnässä täysin korvaamaton työkalu. Analogisten oskilloskooppien hinnat alkavat noin 300 eurosta ja tietokoneeseen liitettävien hinnat alkavat noin 150 eurosta. Koska mikrokontrollereiden nopeus ja ominaisuudet ovat lisääntyneet huimaa vauhtia, on mahdollista rakentaa yksinkertainen oskilloskooppi käyttäen sopivaa mikrokontrolleria sekä vähäistä määrää erilliskomponentteja.

Työn tarkoituksena on rakentaa nopeaa mikrokontrolleria hyödyntävä, tietokoneeseen liitettävä oskilloskooppi. Mikrokontrollerilla suoritetaan mitattavan signaalin muuttaminen digitaaliseen muotoon sekä digitaalisen signaalin lähetys tietokoneelle USB-yhteydellä. Tietokoneella toteutetaan yksinkertainen käyttöliittymä käyttäen Qt-kehitysympäristöä. Myös muu raskaampi laskenta suoritetaan tietokoneella, kuten esimerkiksi FFT-muunnos.

Käytettäväksi mikrokontrolleriksi valittiin STM32F407, jolle Atollic-nimiseltä yritykseltä on saatavana ilmainen, mutta hieman riisuttu kehitysympäristö. Lisäksi käyttöliittymän suunniteluun käytetty Qt on ilmainen, avoimen lähdekoodin, kehitysympäristö. Työn kokonaiskustannukset ovat murto-osa kaupallisen oskilloskoopin hinnasta, tosin myös ominaisuudet ovat hieman vähäisemmät.

2 USB-PROTOKOLLA [1]

USB:n kantavana ideana on liikennöintiprotokollan tarkka määrittely. Tällä saavutetaan se, että jokaisen laitevalmistajan ei tarvitse kirjoittaa erillisiä ajureita PC:lle, vaan se jätetään käyttöjärjestelmän tehtäväksi. Esimerkiksi kaikki muistitikut toimivat samoilla ajureilla, koska ne noudattavat USB:n massamuistilaitteille määriteltyä protokollaa.

USB on palvelin – asiakas-tyyppinen protokolla. Kun laite, eli asiakas, liitetään tietokoneeseen (isäntä), niin isäntä asettaa laitteelle yksilöllisen osoitteen ja pyytää tältä perustiedot. Perustiedoissa laite kertoo isännälle, millainen laite on kyseessä, valmistajan ID:n, tuotteen ID:n, kuinka monta rajapintaa (interface) laitteessa on ja kuinka monta liityntäkohtaa jokaisella rajapinnalla on. Yhteensä erinäisiä tietoja siirtyy useita kymmeniä. Valmistajan ID:n ja tuotteen ID:n perusteella käyttöjärjestelmä osaa laittaa oikeat ajurit hoitamaan liikennöintiä laitteen ja tietokoneen välillä.

USB:n liikennöinti on pakettipohjaista. Jokainen paketti koostuu tietyistä kentistä. Kaikki paketit alkava Sync-kentällä, jota käytetään lähettäjän ja vastaanottajan kellojen synkronointiin. PID-kenttä kertoo, millainen paketti on kyseessä: merkki-, data-, kättely- tai erikoispaketti. ADDR-kentässä on laitteen osoite, jolle paketti on tarkoitettu. Mahdollisia osoitteita on 127. Jokaisen laitteen osoite on nolla siihen asti, kunnes isäntä asettaa laitteelle uuden osoitteen. ENDP-kenttä kertoo, mihin laitteen liityntäpisteeseen paketti on osoitettu. CRC-kentässä siirretään hyötykuorman tarkiste. Merkkipaketeille käytetään 5-bittistä tarkistetta ja datapaketeille 16-bittistä tarkistetta. Jokainen paketti loppuu EOP-kenttään.

Erilaisia pakettityyppejä on neljä. Merkkipaketti kertoo, millainen sitä seuraava paketti on tyypiltään. IN-merkkipaketti kertoo laitteelle, että palvelin on valmis vastaanottamaan dataa. OUT-merkkipaketti kertoo, että palvelin aikoo lähettää laitteelle dataa. Datapaketti sisältää nimensä mukaisesti siirrettävän data. Sitä edeltää aina merkkipaketti, joka kertoo, minne datapaketti on menossa. Kättelypaketilla ilmoitetaan tiedonsiirron tila. ACK-kättelypaketilla kuitataan edellinen paketti onnistuneesti vastaanotetuksi. NAK-kättelypaketilla laite ilmoittaa palvelimelle, että se ei pysty tällä hetkellä lähettämään tai vastaanottamaan dataa. STALL-kättelypaketilla laite ilmoittaa isännälle, että

laitteessa on vikatilanne. USB-liikenne on jaettu noin millisekunnin mittaisiin ikkunoihin. Palvelin tiedottaa jokaisen uuden ikkunan alkamisesta SOF-paketilla.

USB:ssä on neljä erilaista tiedonsiirtotapaa: ohjausviestien siirto, massasiirto, isokroonninen siirto ja keskeyttävä siirto. Ohjausviestien siirto on nimensä mukaan varattu laitetta ohjaavien komentojen siirtoon, joten sen nopeus ei ole kovin suuri. Tällä tavalla tapahtuu tiedonsiirto, kun laite ensimmäisen kerran kytketään tietokoneeseen. Jokaisessa USB-laitteessa on oltava vähintään yksi tällainen kanava. Pakettitasolla tiedonsiirron aloittaa aina palvelin. Kun palvelin haluaa lukea laitteelta tietoa, lähettää palvelin IN-paketin. Tähän laitteen on vastattava joko datapaketilla tai NAK-paketilla, mikäli tietoa ei tällä hetkellä ole luettavaksi. Palvelin kuittaa paketin vastaanotetuksi ACK-paketilla. Vastaavasti kun palvelin haluaa kirjoittaa laitteelle tietoa, lähettää palvelin ensimmäisenä OUT-paketin ja heti perään itse datapaketin. Laite kuittaa paketin vastaanotetuksi ACK-paketilla, tai jos laite ei ehdi juuri sillä hetkellä lukemaan datapakettia, niin se vastaa NAK-paketilla.

Massasiirtoa käytetään yleisesti suurten tiedostojen ja muun purskeina esiintyvän tiedon siirtoon. USB:n kapasiteetista 10 prosenttia on varattu ohjausviestien siirtoon. Loput 90 prosenttia ovat ensisijaisesti varattu isokroonista ja keskeyttävää tiedonsiirtotapaa käyttäville laitteille. Massasiirto siirtää tietoa, kun USB-väylä ei ole muiden laitteiden käytössä. Tätä siirtotapaa käytetään yleisesti esimerkiksi muistitikuissa. Pakettitasolla tiedonsiirto tapahtuu samalla tavalla kuin ohjausviestien siirrossa.

Isokroonisessa siirtotavassa laitteelle taataan suurin mahdollinen siirtonopeus. Siirretylle tiedolle tehdään virheentarkistus, mutta virheen sattuessa pakettia ei lähetetä uudelleen. Siirtotapaa käytetään sovelluksissa, joissa aikakriittistä tietoa siirretään jatkuvana virtana, mutta sillä ei ole suurta merkitystä, vaikka välillä tapahtuisikin tiedon korruptiota. Tällaisia sovelluksia ovat esimerkiksi äänentoisto ja videontoisto. Pakettitasolla tiedonsiirto eroaa aiemmista. Palvelin lähettää tasaisin aikavälein IN-paketin, johon laite vastaa datapaketilla. Jos palvelin haluaa lähettää laitteelle tietoa, lähettää se ensin OUT-paketin ja seuraavana datapaketin. Paketteja ei siis kuitata vastaanotetuiksi, eikä paketteja näin ollen myöskään lähetetä uudelleen, jos ne katoavat matkalla.

Keskeyttävä tiedonsiirtotapa on samankaltaista kuin isokrooninen. Palvelin lähettää jaksollisesti IN-paketin, johon laite vastaa joko datapaketilla, NAK-paketilla tai STALL-paketilla. Vastaavasti toimitaan, kun laite haluaa lähettää tietoa palvelimelle. Toisin kuin isokroonisessa siirtotavassa, keskeyttävässä siirtotavassa kyselyiden väliaika voidaan määrittää laitteen asetuksissa. Tätä käytetään yleisesti niin sanotuissa HID-laitteissa (Human Inteface Device), joita ovat käytännössä erilaiset ohjaimet, hiiret ja näppäimistöt.

Kun laite ensimmäisen kerran kytketään tietokoneeseen, käyttöjärjestelmä pyytää laitteelta laitekuvauksen. Laitekuvaus sisältää muun muassa laitteen luokan ja alaluokan (esim. massamuisti), siirrettävän datapaketin koon, valmistajan sarjanumeron sekä tuotteen sarjanumeron. Näiden tietojen perusteella käyttöjärjestelmä osaa liittää laitteeseen oikeat ajurit.

Seuraavaksi käyttöjärjestelmä pyytää asetuskuvauksen ja rajapintakuvauksen. Asetuskuvauksessa kerrotaan, montako rajapintaa laitteessa on ja paljonko virtaa laite tarvitsee USB-portista. Rajapintakuvaus sisältää tiedon siitä, millainen rajapinta on kyseessä ja montako liityntäpistettä siinä on. Samassa laitteessa voi olla useita rajapintoja: esimerkiksi puhelimet esiintyvät usein massamuisteina, mutta tarjoavat myös internet-yhteyden jaon saman USB-kaapelin läpi.

Jokaisesta liityntäpisteestä lähetetään myös kuvaus. Kuvauksessa kerrotaan liityntäpisteen osoite, tiedonsiirtotapa ja kerralla siirrettävän paketin suurin sallittu koko. Osoitteen vähiten merkitsevä bitti osoittaa, mihin suuntaan liityntäpiste siirtää tietoa. Kun kaikki laitteen asetuksen on saatu luettua ja ajurit asennettua, on laite käyttövalmis ja siihen voidaan yhdistää sopivalla ohjelmalla.

3 AD-MUUNNOS

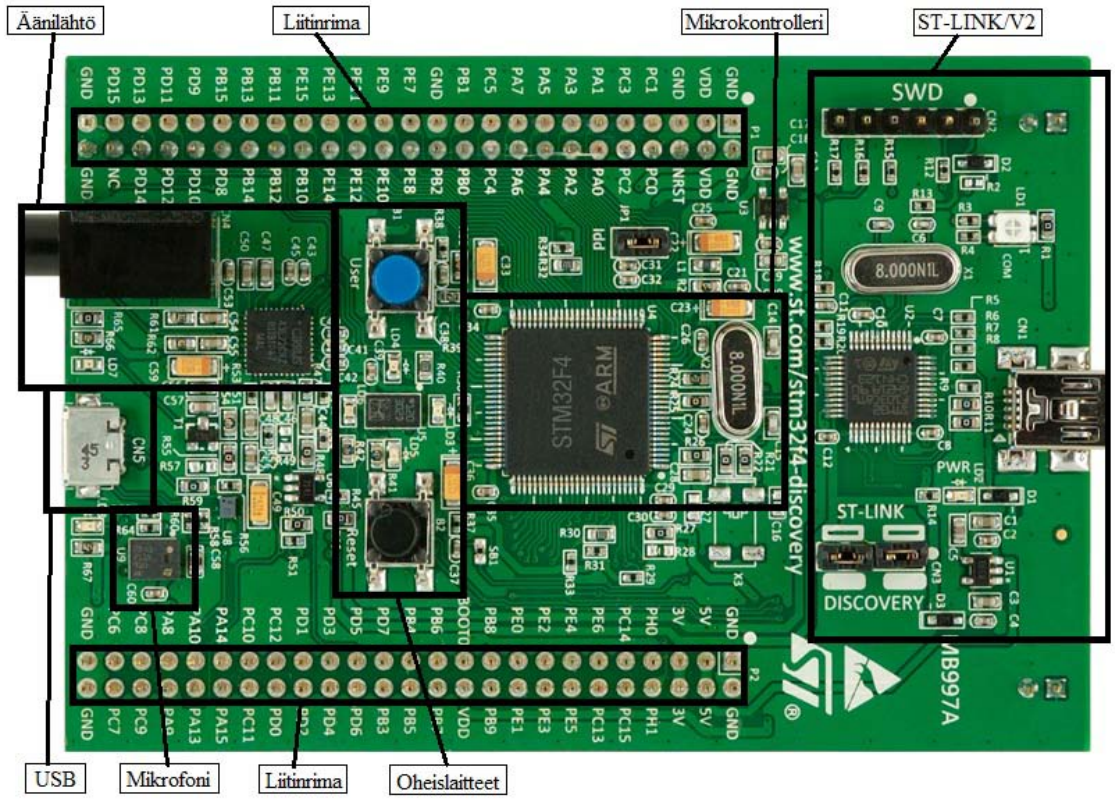
Kun analoginen signaali halutaan muuttaa digitaaliseen muotoon, tarvitaan tämän tekemiseen AD-muunninta. Muuntimelle tuodaan signaali, joka halutaan digitaaliseen muotoon. Signaalia verrataan tunnettuun referenssijännitteeseen ja siitä otetaan näyte. Näyte edustaa signaalin jännitettä tietyllä hetkellä. Kun näytteitä otetaan riittävän suurella taajuudella, voidaan niistä muodostaa analogista signaalia edustava lukujono.

Riittävä näytteenottotaajuus on Nyquist-Shannon teoreeman mukaan kaksi kertaa analogisen signaalin suurin taajuus. Signaalin ja näytteistyksen vaihe-erosta johtuen on mahdollista, että näytteet sattuvat osumaan signaalin nollakohtiin, joten signaalin rekonstruoiminen on mahdotonta. Käytännössä onkin syytä käyttää hieman suurempaa näytteistystaajuutta kuin teoreeman mukaan olisi tarpeellista. [2]

Signaalista otetaan näyte pitopiirillä. Piiri lataa kondensaattoria signaalilla ennalta määrätyn ajan. Tämän jälkeen kondensaattoriin varautunutta jännitettä verrataan tunnettuun referenssijännitteeseen ja muutetaan se numeroksi. Numeroarvo vaihtelee välillä $0 – 2^n$-1, missä n on muunnosresoluutio bitteinä. Suurin arvo vastaa referenssijännitettä ja pienin negatiivista referenssijännitettä, yleensä maatasoa.

Koska digitoitu signaali voidaan ilmaista äärellisellä määrällä arvoja, syntyy AD-muunnoksessa aina virhettä. Tätä virhettä kutsutaan kvantisointikohinaksi. Signaali -kohina- suhde voidaan laskea kaavasta $SQNR = 20 * \log(2^n)$, missä n on muunnosresoluutio bitteinä. Kaava pitää paikkansa vain sinimuotoiselle signaalille, mutta antaa jonkinlaista tuntumaa siitä, miten bittilukumäärä vaikuttaa kohinan määrään. Esimerkiksi 12 bitillä SQNR = 72 dB kun taas 6 bitillä SQNR on 36 dB. [3]

4 MIKROKONTROLLERI



Kuva 1: STM32F4-DISCOVERY kehitysalusta [4]

Työssä käytettiin STM32F4-DISCOVERY kehitysalustaa, jossa on STM32F407VG-mikrokontrolleri. Kehitysalusta on esitetty kuvassa 1 ja kehitysalustan kytkentäkaavio on liitteessä 1. Liitteen ensimmäisellä sivulla on kehitysalustan eri lohkot: ST-LINK/V2, mikrokontrolleri, USB, audio ja oheislaitteet. ST-LINK/V2-lohkon kytkentäkaavio on esitetty tarkemmin liitteen toisella sivulla. ST-LINK/V2 on kehitysalustaan integroitu ohjelmointilaite ja debuggeri. Poistamalla jumpperit liittimestä CN3, voidaan laitetta käyttää ulkoisten mikrokontrollereiden ohjelmointiin. ST-LINK/V2-lohkossa on myös käyttöjännitteiden regulointi.

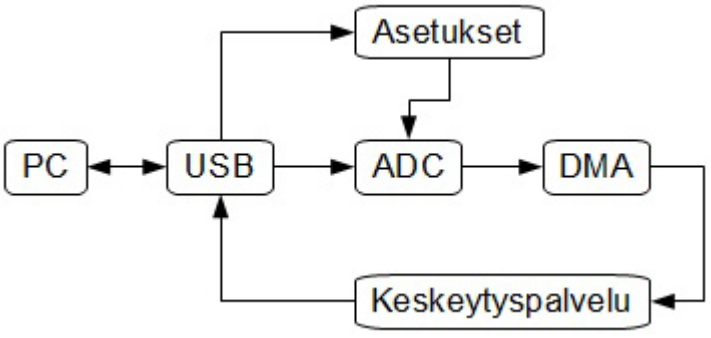
Liitteen kolmannella sivulla on mikrokontrollerin jännitekytkennät ja kellokiteiden kytkennät. Kahdeksan megahertsin kiteestä tuotetaan mikrokontrollerin sisäisellä PLL-piirillä prosessorin kellosignaali, jonka taajuus on tämän työn yhteydessä 168 MHz. 32,768 kHz:n kiteellä tuotetaan reaaliaikakellon tarvitsema kellosignaali. Liitteen neljännellä sivulla on äänilähdön ja mikrofonin kytkentäkaavio.

Liitteen viidennellä sivulla on USB-yhteyden kytkentäkaavio. Kaaviosta näkyy, että mikrokontrolleri on suojattu pieniltä staattisilta purkauksilta mikropiirillä EMIF02-USB03F2. Mikrokontrolleri voidaan ohjelmoida toimimaan myös USB-isäntänä. Tällöin sen pitäisi mahdollisesti antaa virtaa asiakaslaitteelle. Asiakaslaitteessa esiintyvän oikosulun varalta virransyöttö on rajoitettu mikropiirillä STMPS2141STR. Kaaviossa ei näy USB-standardin vaatimaa ylösvetovastusta DP-linjassa. Ylösvetovastuksena käytetään mikrokontrollerin IO-portissa olevaa ohjelmoitavaa vastusta. Liitteen kuudennella sivulla on oheislaitteiden kytkentäkaaviot. Oheislaitteita ovat reset-nappi, ohjelmoitava nappi, neljä merkkilediä ja kolmiakselinen kiihtyvyysanturi.

Kun piiriin kytketään käyttöjännite, jää ohjelma odottamaan USB-yhteyden muodostumista. Kun yhteys on muodostettu, jäädään odottamaan toimintakomentoja. Mahdollisia komentoja ovat asetusten muuttaminen ja AD-muunnoksen aloittaminen. Mikrokontrollerin pääohjelman lähdekoodi on liitteessä 2.

Kun piiri saa komennon aloittaa AD-muunnos, aloitetaan muunnosten tekeminen ja muunnostulosten siirto DMA:lle. Kun yksi muunnos on saatu valmiiksi, lähtee DMA:lle automaattisesti pyyntö lukea muunnettu arvo. Kun arvoja on luettu puskuriin mahtuva määrä, DMA keskeyttää. Keskeytyspalvelun lähdekoodi on liitteessä 3.

Keskeytyspalveluun tultaessa pysäytetään AD-muunnin. Tämän jälkeen muunnostulokset muutetaan edelleen millivolteiksi ja lähetetään eteenpäin USB-ajureille. Ajurit lähettävät tulokset itsenäisesti tietokoneelle. Lopuksi jäädään odottamaan seuraavaa käskyä tietokoneelta. Mikrokontrollerin toiminnallinen lohkokaavio on esitetty kuvassa 2.



Kuva 2: Mikrokontrollerin tominnallinen lohkokaavio

4.1 AD-muunnos

Mikrokontrollerissa on kolme AD-muunninta. Muuntimet voidaan asettaa toimimaan yksitellen tai lomittain. Yksitellen toimiessaan jokainen AD-muunnin käyttäytyy itsenäisenä muuntimena. Lomittaisessa moodissa muuntimet muuntavat yhtä kanavaa samaan aikaan. Näin saavutetaan kolminkertainen näytteistysnopeus yksittäisesti toimivaan muuntimeen verrattuna. Tämän työn tarkoituksiin riittää yhdellä muuntimella saavutettava nopeus.

Mikrokontrollerin sisäiset muistialueet ja oheislaitteet on kytketty toisiinsa tiedonsiirtoväylillä. Väyliä on toistakymmentä, joista ohjelmoijalle käytännössä näkyvät AHB (Advanced High-performance Bus) sekä APB (Advanced Peripheral Bus). AD-muuntimet ovat APB-väylässä, jonka kellotaajuus on tämän työn yhteydessä 84 MHz. Väylän kellotaajuudesta saadaan AD-muuntimien kellotaajuus esijakajan kautta. Esijakajan mahdollisia arvoja ovat 2, 4, 6 ja 8. Näin ollen AD-muuntimen kellotaajuus on 42, 21, 14 tai 10,5 MHz.

Yhteen muunnokseen kuluva aika lasketaan AD-muuntimen kellojaksoina. Yhden näytteen näytteistysaika on ohjelmoitavissa 3 – 480 kellojakson välillä. Lisäksi muunnostarkkuus lisätään suoraan kokonaisaikaan, esimerkiksi kahdeksan bitin muunnostarkkuudella kokonaisaikaan lisätään kahdeksan kellojaksoa. Suurin mahdollinen näytteenottotaajuus saadaan kun muunnostarkkuus on 6 bittiä, näytteistysaika kolme kellojaksoa ja AD -muuntimen kellotaajuus 42 MHz. Näillä arvoilla näytteenottotaajuudeksi tulee 4,67 miljoonaa näytettä sekunnissa. Vastaavasti hitain näytteenottotaajuus on 21341 näytettä sekunnissa.

Hitaampaa näytteenottotaajuutta tarvittaessa AD-muuntimen voi ajastimen avulla ohjelmoida ottamaan yksittäisiä näytteitä tietyin aikavälein, mutta tämän työn yhteydessä todetaan 21 tuhatta näytettä sekunnissa riittävän hitaaksi.

Kun AD-muunnin on saanut yhden muunnoksen valmiiksi, se kopioi tuloksen sille varattuun muistipaikkaan. Tämän jälkeen DMA käy kopioimassa tuloksen talteen taulukkoon. DMA toimii rautatasolla, joten prosessointiaikaa ei kulu muunnostuloksen siirtymisessä AD-muuntimelta taulukkoon. Taulukon täytyttyä DMA generoi keskeytyksen.

Keskeytyspalvelussa ensimmäisenä sammutetaan AD-muunnin. Muunnoksia on turha tehdä jatkuvasti, koska ei ole tarpeen tai aina edes mahdollista esittää muunnostuloksia reaaliaikaisesti. Seuraavaksi muunnostulokset muutetaan millivolteiksi. Tulokset muutetaan kaavalla $adr * V_{ref} / 2^n$, jossa *adr* on muunnostulos, V_{ref} on referenssijännite ja *n* on muunnostarkkuus. Referenssijännite on tässä työssä 3,3 volttia, joten muunnostulokset ovat välillä 0-3300. Kun arvot on muutettu millivolteiksi, taulukko kopioidaan USB:n lähetyspuskuriin.

4.2 USB

Mikrokontrolleria voidaan käyttää USB-standardin mukaisilla siirtonopeuksilla 1,5 Mbps, 12 Mbps ja 480 Mbps. Tässä työssä nopeudeksi riittää 12 Mbps ja lisäksi käytetyssä kehitysalustassa on valmiina liittimet tälle nopeudelle. Laitevalmistajalta on saatavana kyseiselle mikrokontrollerille USB-ajurit, jotka piilottavat suurimmalta osin protokollan sisäisen toiminnan ohjelmoijalta.

USB voidaan ajatella rakentuvan kerroksista. Alimpana kerroksena on rautatason toteutus ja seuraavana matalan tason ajurit. Seuraavassa kerroksessa on tiedonsiirtopisteiden luku ja kirjoitus. Kolmannessa kerroksessa on luokkatoteutus, esimerkiksi HID, CDC tai massamuisti. Ylimpänä on sovelluskerros. Valmistajan ajurit hoitavat ylintä kerrosta lukuun ottamatta kaiken tiedonsiirron. [5]

Tiedonsiirtotavaksi valittiin massatiedonsiirto (Bulk), joka ei toimi täydellä nopeudella mutta sillä varmistetaan, että siirretty tieto ei korruptoidu matkalla. Laite on ohjelmoitu esittäytymään tiedonsiirtoluokan laitteena (CDC, Communication Device Class). Sillä tosin ei ole suurta merkitystä, koska työssä ei voida käyttää geneeristä ohjelmaa tietokoneen puolella. Lisäksi laite kertoo tietokoneelle, että laitteessa on yksi tiedonsiirtokanava ulospäin, yksi sisäänpäin ja yksi kanava ohjauskomennoille.

Toinen mahdollinen tiedonsiirtotapa tähän sovellukseen olisi isokrooninen tiedonsiirto. Se on suunniteltu sovelluksiin, joissa tietoa siirretään jatkuvana virtana, joten sillä periaatteessa voisi siirtää näytteitä lähes reaaliajassa. Koska näytteitä otetaan lyhyissä purskeissa, on massatiedonsiirto järkevämpi valinta.

Massatiedonsiirtotavassa kerralla siirtyvän paketin hyötykuorma on 64 tavua. Kun AD-muuntimelta tulevat tulokset ovat kahdessa tavussa, on ne pätkittävä 8 bitin paloihin ennen USB-ajureille lähettämistä. Paloittelu tapahtuu siten, että USB-puskurin parittomiin paikkoihin kirjoitetaan muunnostuloksen enemmän merkitsevä osa ja parillisiin paikkoihin vähemmän merkitsevä osa. Kun tulokset on paloiteltu, asetetaan osoitin osoittamaan taulukon loppuun ja kutsutaan USB-lähetysfunktiota, joka ei varsinaisesti tee mitään, vaan ilmoittaa ajureille, että dataa olisi lähetettävänä.

Datan lähetyksen lisäksi USB-ajurit hoitavat komentojen vastaanottamisen. Mahdollisia komentoja ovat AD-muuntimen käynnistäminen, muunnosresoluution muuttaminen, näytteistysajan muuttaminen ja AD-muuntimen kellotaajuuden muuttaminen. Komennot koostuvat yhdestä kirjaimesta, joka kertoo, mikä komento on ja sitä seuraa tarvittavat parametrit. Esimerkiksi haluttaessa muuttaa muunnostarkkuus 12-bittiseksi, lähetetään laitteelle merkkijono ”W12”. Kun komento on vastaanotettu, laite pysäyttää AD-muuntimen ja käynnistää sen uudelleen uusilla asetuksilla. Lähetyksen ja vastaanoton lähdekoodi on liitteessä 4.

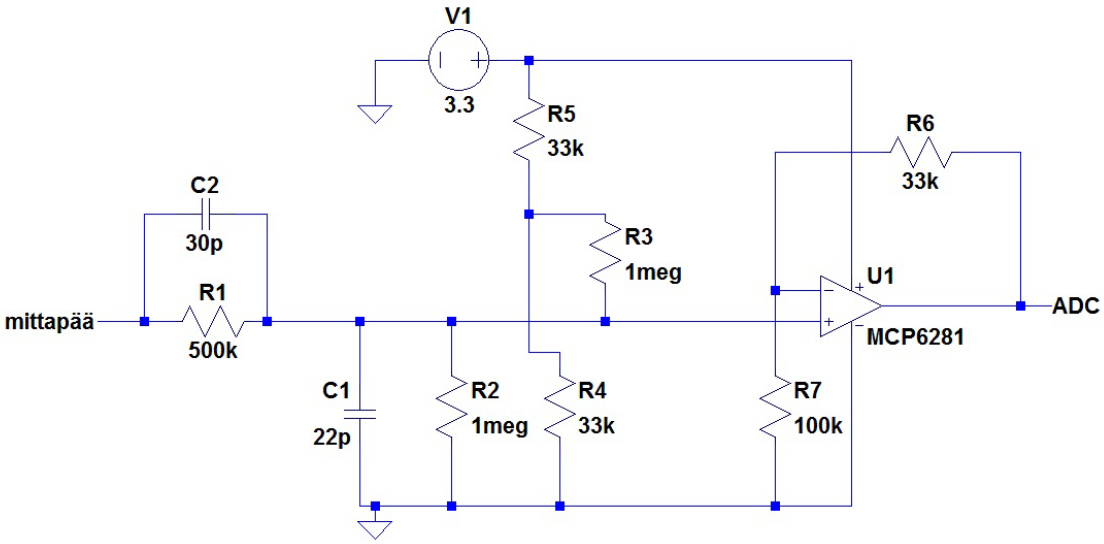
4.3 DMA

Direct Memory Acces on mikrokontrollerin ominaisuus, jolla voidaan siirtää tietoa kahden muistipaikan välillä automaattisesti. Käskyn tullessa DMA käy lukemassa sille osoitetusta muistipaikasta datan rekisteriin. Rekisteristä data kopioidaan sille osoitettuun lopulliseen muistipaikkaan. Lopuksi DMA kasvattaa laskuria, jolla se seuraa, kuinka monta siirtoa on vielä tehtävä.

DMA voi kasvattaa lähde- tai kohdemuistipaikan osoitetta joka lukukerralla, joten tietoa voidaan siirtää taulukoina. Tässä työssä DMA on ohjelmoitu lukemaan AD-muuntimen tuloksia tietystä osoitteesta ja kopioimaan ne kohdetaulukkoon. Jokaisen yksittäisen muunnostuloksen valmistuttua AD-muunnin lähettää automaattisesti lukupyynnön DMA:lle, joten tähän ei kulu prosessoriaikaa. Kun tuloksia on luettu ennalta määrätty määrä, DMA generoi keskeytyksen, jossa data voidaan prosessoida.

4.4 AD-tulovahvistin

Mittapäänä haluttiin käyttää tavallista oskilloskoopin mittapäätä, joten rakennettiin kytkentä jolla saatiin mittapää liitettyä mikrokontrolleriin. Kytkennän pääasiallinen tehtävä on muodostaa noin yhden megaohmin resistanssi, jotta mittapään 10-kertainen vaimennus toimisi oikein. Lisäksi kytkentä summaa mitattavaan signaaliin pienen määrän tasajännitettä, että voidaan mitata myös hieman negatiiviselle puolelle meneviä signaaleja. Mikrokontrollerissa käytettävät matalat jännitteet rajoittavat tulovahvistimena käytettävän operaatiovahvistimen valintaa. Operaatiovahvistimen on oltava rail-to-rail-tyyppiä, eli sen tulo- ja lähtöjännitteet voivat olla muutaman millivoltin päässä käyttöjännitteistä. Piirin on myös toimittava 3,3 voltin yksipuolisella käyttöjännitteellä. Lisäksi piirin taajuuskaista olisi hyvä olla kohtalaisen suuri. Operaatiovahvistimeksi valittiin MCP6281. Kyseinen piiri täyttää edellä mainitut kriteerit ja sen kaistanleveys on viisi megahertsiä vahvistuksen ollessa yksi. Vahvistuksen kasvaessa kaistanleveys pienenee.



Kuva 3: AD-tulovahvistimen kytkentä

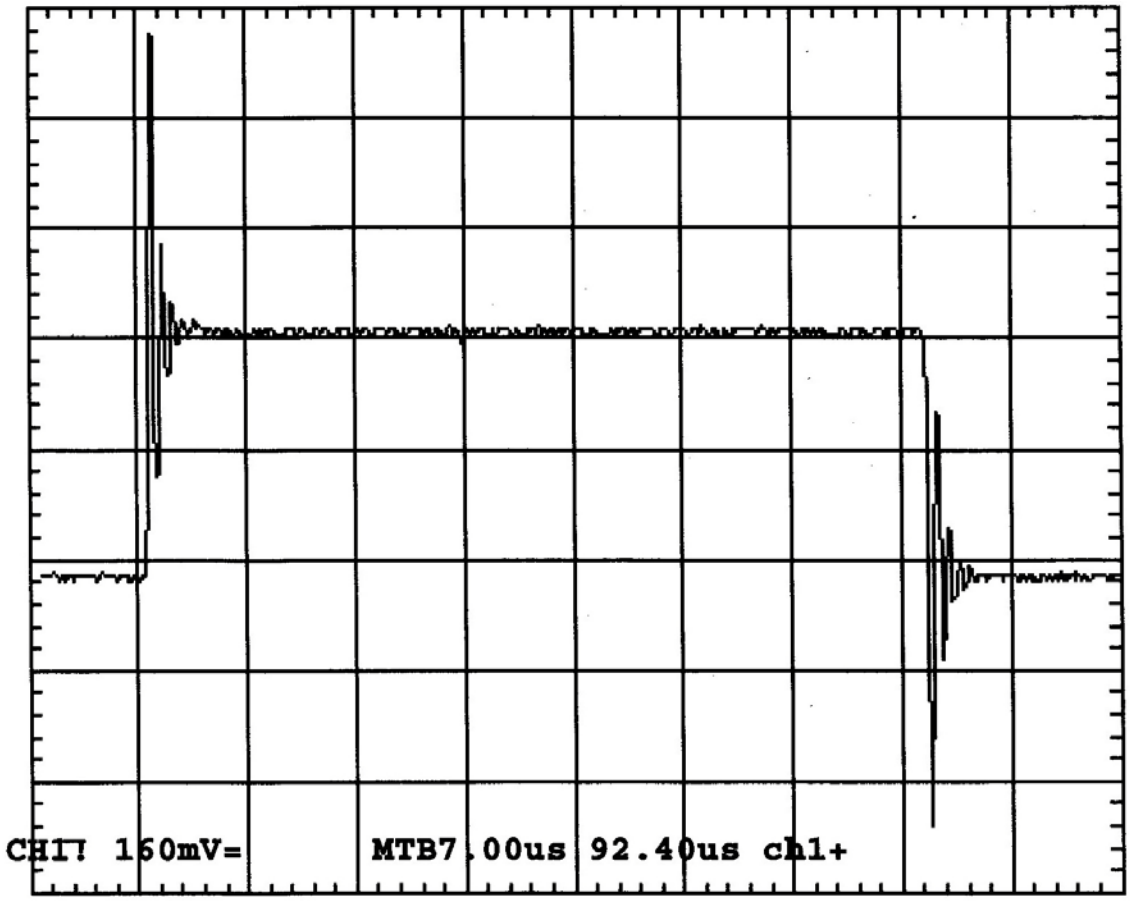
Kuvassa 3 on esitetty tulovahvistimen kytkentäkaavio. Vastusten R2 ja R3 rinnankytkennästä tulee puolen megaohmin resistanssi, lisäksi näiden kanssa sarjassa on vastus R1, joten tulosta näkyvä resistanssi on yksi megaohmi. Tosiasiassa vastukset R4 ja R5 nostavat vastusta hieman, mutta niin vähän, että ne voidaan jättää huomiotta. Vastuksista R1 ja R2 || R3 muodostuu jännitteenjako, joten signaalista viedään operaatiovahvistimelle puolet. Suuremmilla taajuuksilla samainen jännitteenjako tapahtuu kondensaattorin C2, C1 ja muun kytkennästä aiheutuvan kapasitanssin välillä. Koska kytkennän

todellista kapasitanssia ei tiedetä tarkasti, on kondensaattori C2 todellisessa kytkennässä 22 pF ja muutaman pikofaradin säätökondensaattorin rinnankytkentä. Näin voidaan säätää C2:n arvo vastaamaan muun kytkennän kapasitanssia. Takaisinkytketyn operaatiovahvistimen vahvistus on 1 + R6 / R7 = 4/3. Tulovaimennus huomioon ottaen koko kytkennän vahvistukseksi tulee 2/3.

Käytettäessä mittapään 10x-asentoa, kytkeytyy vastuksen R1 kanssa sarjaan yhdeksän megaohmin vastus. Mittapäästä ja muusta kytkennästä muodostuvan jännitteenjaon jälkeen signaalista on jäljellä 1/20. Koska operaatiovahvistimen vahvistus pysyy samana, on kokonaisvahvistus siten 1/15.

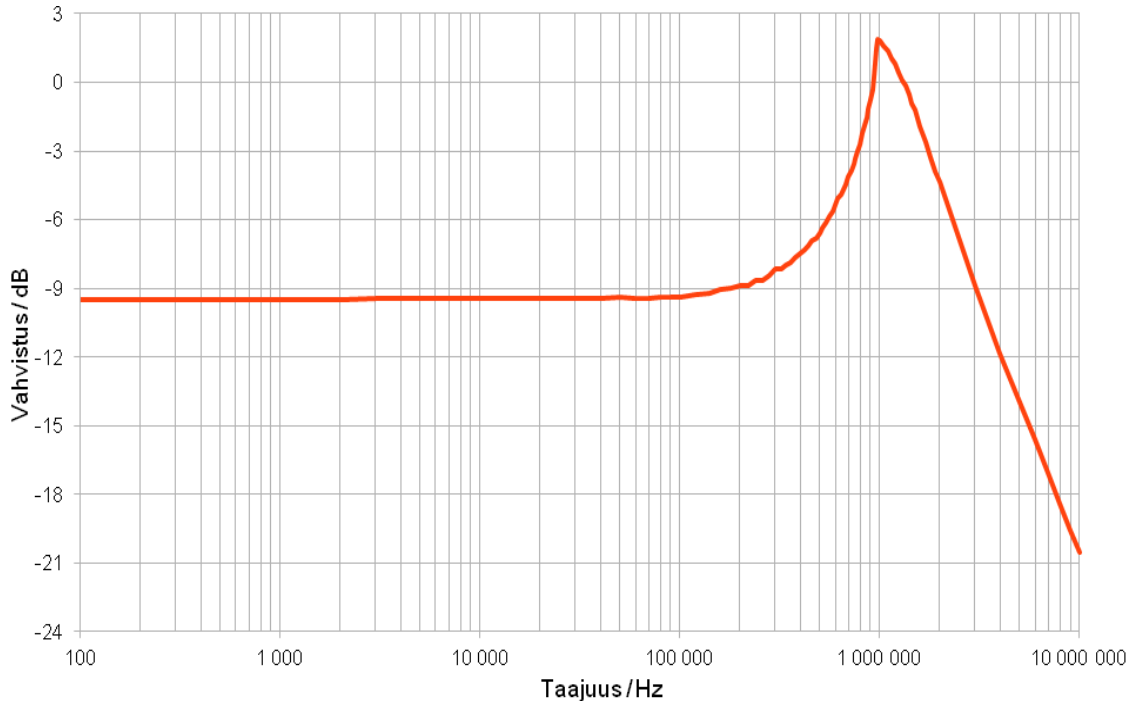
Kytkennässä on väkisin hieman kapasitanssia. Mittapään ja muun kytkennän kapasitanssien suhde on oltava sama kuin resistanssien suhde kääntäen. Piirilevystä aiheutuu arviolta 5 pF kapasitanssia ja operaatiovahvistimen tulokapasitanssi on 6 pF. Oskilloskoopin mittapäissä on yleisesti muutaman pikofaradin säädettävä kondensaattori, jolloin mittapään ja kytkennän kapasitanssien suhteeksi saadaan 1/9.

Kytkennän askelvaste mitattiin oskilloskoopilla ja signaaligeneraattorilla. Kytkentään syötettiin kanttiaaltoa signaaligeneraattorista ja ulostulo mitattiin oskilloskoopilla. Signaalin nousuajaksi mitattiin noin 150 nanosekuntia. Asettumisajaksi mitattiin 2 mikrosekuntia. Ylitystä ja alitusta esiintyy noin 185%. Tulos on esitetty kuvassa 4.



Kuva 4: AD-tulovahvistimen askelvaste

Kytkennän taajuusvaste mitattiin oskilloskoopilla ja signaaligeneraattorilla. Kytkentään syötettiin signaaligeneraattorista siniaaltoa, jonka jännite oli 500 mVpp. Signaalin jännite mitattiin kytkennän lähdöstä oskilloskoopilla. Kytkentään syötettävän signaalin taajuutta kasvatettiin 100 hertsistä kymmeneen megahertsiin, ottaen kymmenen arvoa dekadia kohden. 100 kHz:stä yhteen megahertsiin otettiin mittaustulos 20 kHz:n välein. Tulokset on esitetty kuvassa 5.



Kuva 5: AD-tulovahvistimen taajuusvaste

Kytkennän kolmen desibelin taajuus on noin 500 kHz, jolloin signaalin jännite on kasvanut √2-kertaiseksi. Kuuden desibelin taajuus on 750 kHz, jolloin signaalin jännite on kasvanut kaksinkertaiseksi. Kun kytkennän vahvistus kasvaa liikaa, nousee signaalin huippujännite suuremmaksi kuin operaatiovahvistimen positiivinen käyttöjännite ja vastaavasti signaalin negatiivinen huippujännite tippuu alle operaatiovahvistimen negatiivisen käyttöjännitteen. Koska signaaliin summataan alle puolet käyttöjännitteestä, leikkautuu signaalin negatiivisen puoliskon huippu ennen positiivista huippua. Tästä aiheutuu kuvaajan huipun terävyys. Kolmen ja kuuden desibelin taajuuksiin tällä ei ole vaikutusta ennen kuin signaalin jännite kasvaa niin suureksi, että leikkautuminen tapahtuu jo kolmen tai kuuden desibelin vahvistuksella.

5 PC-PUOLEN TOTEUTUS

5.1 USB-ajurit [6]

Tietokoneen puolella USB-ajureina käytettiin libusb-win32-kirjastoa [7]. Paketin mukana tulee ohjelma, jolla generoidaan laitteelle ajurit. Kun ajurit on asennettu normaaliin tapaan, käyttöjärjestelmä tunnistaa laitteen. Paketin mukana tulee myös C-kielinen ohjelmointirajapinta, jonka avulla laitteen kanssa voidaan keskustella. Kirjasto sisältää USB-siirtoprotokollia, joista tässä työssä käytetään massatiedonsiirtoa.

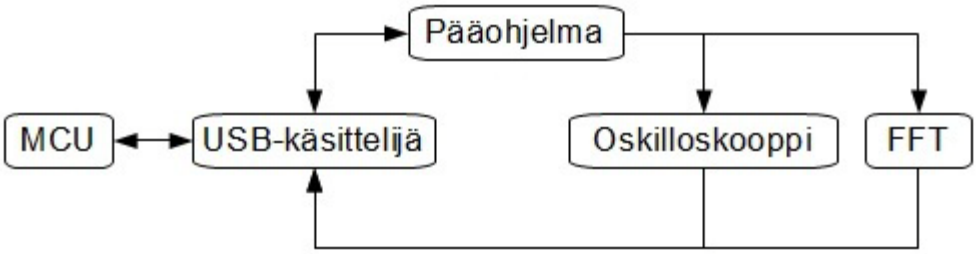
Yhteys voidaan toteuttaa joko synkronisesti tai asynkronisesti. Kun ohjelmakoodissa ajetaan asynkroninen lukufunktio, ajettu funktio palauttaa kontrollin välittömästi kutsuneelle koodille. Ajurit suorittavat lukuoperaation taustalla ja luettu tieto on funktiolle parametrina annetussa muistipaikassa jonkin ajan päästä. Laitteeseen kirjoitetaan vastaavalla tavalla. Tämän etu on se, että luku/kirjoitustapahtuma ei turhaan hidasta pääohjelmaa odottamalla luku/kirjoitustapahtuman valmistumista.

Synkroniset luku- ja kirjoitusfunktiot jäävät aina odottamaan tapahtuman valmistumista. Ne ovat hieman yksinkertaisempia käyttää, koska funktiokutsua seuraavassa käskyssä voidaan olla varmoja siitä, että tiedonsiirtoa on vähintään yritetty. Synkronisille kutsuille annetaan parametreina kirjoitusosoite (määrää, onko kyseessä luku- vai kirjoitustapahtuma), puskurin osoite, puskurin koko ja kuinka pitkään korkeintaan odotetaan tapahtuman päättymistä. Tässä työssä käytetään synkronista siirtotapaa.

5.2 Ohjelmisto

Käyttöliittymä tehtiin Qt-kehitysympäristössä, joka on suunniteltu graafisten käyttöliittymien kehitykseen. Qt:n on alun perin kehittänyt norjalainen Trolltech, jonka Nokia osti vuonna 2008. Trolltech jatkaa edelleen Qt:n kehittämistä Nokian alaisuudessa nimellä Qt Development Frameworks. Kehitysympäristö on alustariippumaton ja se on julkaistu LGPL-lisenssin alaisena. Kehitysympäristössä on käytettävissä useita eri ohjelmointikieliä, joista tämän työn toteutukseen valittiin C++.

Yksi Qt:n keskeisiä ominaisuuksia on signaali-vastaanottaja-systeemi (signal-slot). Ajatuksena on, että ohjelman eri moduulit voivat lähettää ja vastaanottaa signaaleja. Signaalin lähettäjä ja vastaanottaja sidotaan yhteen. Kun jokin moduuli lähettää signaalin, niin vastaanottava moduuli ajaa sen perusteella funktion, jossa tehdään jotain tapahtumaan liittyvää. Esimerkiksi tässä työssä USB-käsittelijä lähettää pääohjelmalle signaalin, kun käsittelijä on lukenut laitteelta tietyn määrän muunnostuloksia. Tämän perusteella pääohjelma osaa käydä lukemassa USB-käsittelijältä luetut arvot. Ohjelman toiminnallinen lohkokaavio on esitetty kuvassa 6.



Kuva 6: Ohjelmiston toiminnallinen lohkokaavio

5.2.1 USB-käsittelijä

USB-käsittelijä on vastuussa tiedonsiirrosta laitteen ja tietokoneen välillä. Kun pääohjelmalta tulee käsky yhdistää laitteeseen, niin käsittelijä aloittaa yhteyden alustuksen. Ensimmäisenä luetaan kaikkien tietokoneeseen liitettyjen laitteiden kuvaukset. Laitteita käydään läpi, kunnes löydetään laite, jonka sarjanumerot täsmäävät etsittyyn laitteeseen. Kun etsitty laite on löydetty, se varataan ohjelman käyttöön.

Kun yhteys on muodostettu, lähetetään laitteelle käsky aloittaa ensimmäinen AD muunnos. Samalla viritetään ajastin hälyttämään millisekunnin päästä. Kun ajastin hälyttää, kutsutaan funktiota, joka käy lukemassa muunnostulokset laitteelta. Lukemisen valmistuttua lähetetään pääohjelmalle signaali, jonka perusteella pääohjelma käy lukemassa tulokset USB-käsittelijältä. Tämän jälkeen käsittelijä jää odottamaan seuraavaa komentoa. USB-käsittelijän lähdekoodi on liitteessä 5.

5.2.2 Oskilloskooppi-ikkuna

Oskilloskoopin kuvaajan piirtämisestä huolehtivalla luokalla on kaksi tehtävää. Se seuraa hiiren tapahtumia ja piirtää muunnostuloksista kuvaajan. Pääohjelmalta saadaan tieto siitä, kuinka montaa millivolttia yhden pikselin korkeus edustaa sekä siitä, että kuinka montaa sekuntia yhden pikselin leveys vastaa. Kun lisäksi tunnetaan näytteenottotaajuus, voidaan tarkasti määrittää, mihin kohtaan näyttöä jokainen peräkkäinen muunnostulos on piirrettävä.

Kun oskilloskoopin piirtoikkunaa klikataan hiiren napilla, Qt kutsuu automaattisesti ikkunan luokassa olevaa funktiota, joka käsittelee hiiren klikkauksen. Jos klikkaus oli hiiren vasen nappi, otetaan muistiin klikkauskohdan koordinaatit, joita käytetään myöhemmin, kun lasketaan kahden pisteen erotus aikana ja millivoltteina. Mikäli klikkaus tapahtui hiiren oikealla napilla, lasketaan mitä millivolttiarvoa kyseinen pikseli edustaa ja käytetään tätä arvoa oskilloskoopin liipaisukohtana.

Luokka seuraa myös hiiren liikkumista piirtoikkunan päällä. Aina kun hiiri siirtyy, Qt kutsuu automaattisesti luokan funktiota, joka käsittelee hiiren liikkumisen. Funktiossa lasketaan aiemmin hiiren vasemmalla napilla valitun pikselin ja hiiren nykyisen kohdan erotus. Pisteiden x-koordinaattien erotuksesta lasketaan pisteiden välinen ero aikana ja pisteiden y-koordinaateista lasketaan erotus jännitteenä. Luokan lähdekoodi on liitteessä 6.

5.2.3 FFT-luokka [8]

Spektrianalyysissä aikatason signaali on muutettava taajuustason esitykseksi. Aikatasosta taajuustasoon siirrytään yleisesti Fourier muunnoksella. Diskreetti Fourier-muunnos lasketaan kaavalla:

$$F_n = \sum_{k=0}^{N-1} f_k e^{-i\frac{2\pi}{N}kn} \qquad (1)$$

Yhtälössä 1 N on näytteiden lukumäärä ja f_k on signaalin arvo kohdassa k. Yhtälöllä 1 voidaan laskea Fourier-muunnos kahdella sisäkkäisellä silmukalla. Tällöin algoritmin aikavaativuus on O(N^2), eli kun näytteiden lukumäärä kasvaa kaksinkertaiseksi, Fourier-muunnokseen käytetty laskenta-aika kasvaa nelinkertaiseksi. Kahdeksan näytteen pituiselle signaalille yhtälö voidaan kirjoittaa muotoon:

$$F_n = f_0 + f_1 e^{-i\frac{2\pi}{8}n} + f_2 e^{-i\frac{2\pi}{8}2n} + f_3 e^{-i\frac{2\pi}{8}3n} + f_4 e^{-i\frac{2\pi}{8}4n} + f_5 e^{-i\frac{2\pi}{8}5n} + f_6 e^{-i\frac{2\pi}{8}6n} + f_7 e^{-i\frac{2\pi}{8}7n} \quad (2)$$

Jakamalla signaali parillisiin ja parittomiin termeihin ja ottamalla eksponenttitermejä yhteisiksi tekijöiksi, voidaan yhtälö 2 kirjoittaa muotoon:

$$F_n = \left[\left(f_0 + f_4 e^{-i\pi n}\right) + e^{-i\frac{\pi}{2}n}\left(f_2 + f_6 e^{-i\pi n}\right)\right] + e^{-i\frac{\pi}{4}n}\left[\left(f_1 + f_5 e^{-i\pi n}\right) + e^{-i\frac{\pi}{2}n}\left(f_3 + f_7 e^{-i\pi n}\right)\right] \quad (3)$$

Suurin nopeuslisäys FFT-algoritmiin (Fast Fourier Transform) saadaan kun käytetään hyväksi eksponenttifunktion jaksollisuutta:

$$e^{i(\phi+2\pi)} = e^{i\phi} \text{ ja } e^{i(\phi+\pi)} = -e^{i\phi} \quad (4)$$

Yhtälössä 3 sulkeiden sisällä olevat eksponenttitermit ovat samoja parillisilla n:n arvoilla ja parittomilla n:n arvoilla. Parilliset ja parittomat termit ovat lähes samoja, vain etumerkki vaihtuu. Yhtälössä 3 hakasulkeiden sisällä olevan eksponenttitermin jaksollisuus on neljä. Toisin sanoen termi on sama kun n = 0 tai 4, n = 2 tai 6, n = 1 tai 5 ja n = 3 tai 7. Myös nämä parit käyttäytyvät jaksollisesti. Eksponenttitermin arvo on 1 kun n = 0 tai 4, -1 kun n = 2 tai 6, -i kun n = 1 tai 5 ja i kun n = 3 tai 7. Algoritmi tunnetaan nimellä Cooley – Tukey FFT-algoritmi, jonka aikavaativuus on O($N \log_2 N$).

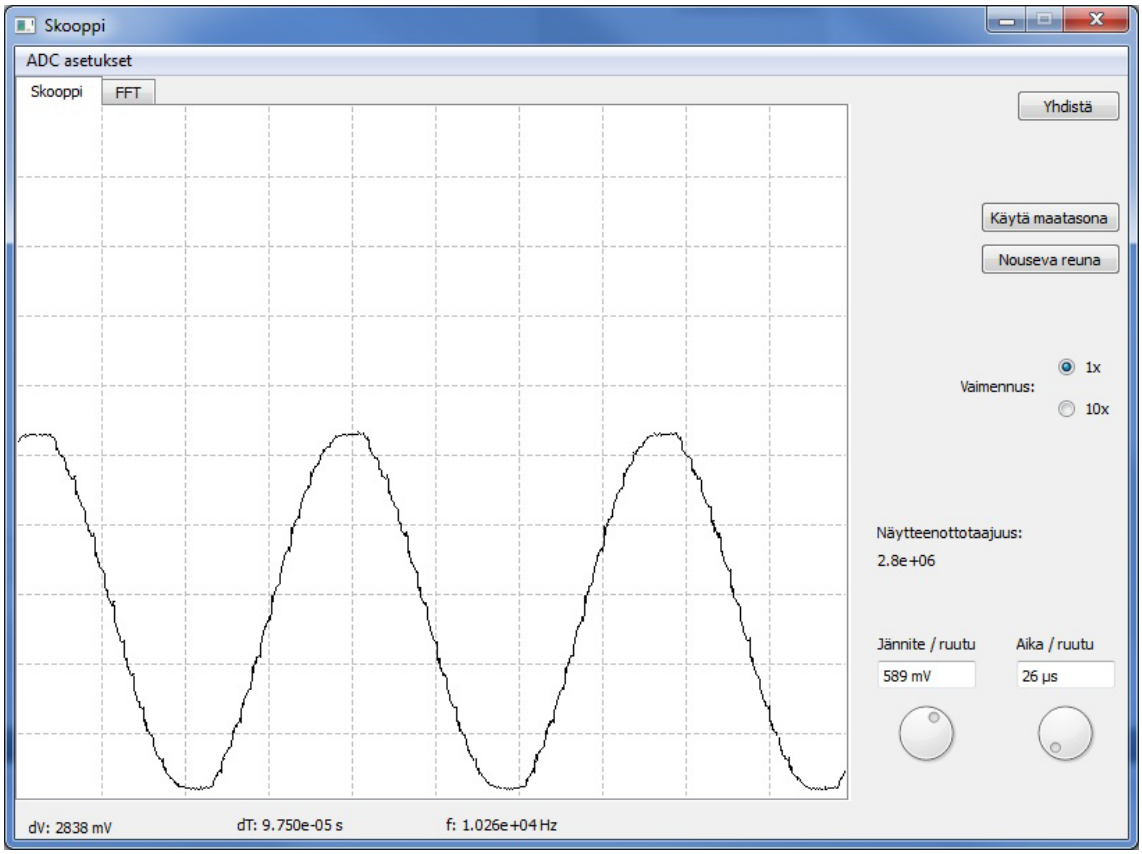
Työssä käytetään Sergey Chernenkon tekemää FFT-kirjastoa [9]. Ohjelman FFT-luokalle annetaan pääohjelmasta signaalin jännitearvot. Signaali FFT-muunnetaan. Muunnostuloksena saadaan taulukko, jossa on signaalin taajuustasoiset arvot. Taulukon puolivälissä on näytteenottotaajuuden Nyqvistin taajuus. Loppuosassa taulukkoa spektri toistuu peilikuvana, joten näytölle piirretään vain puolet taulukosta. Arvot muutetaan ennen piirtämistä desibeleiksi. Luokan lähdekoodi on liitteessä 7.

5.2.4 Pääohjelma

Pääohjelmassa edellä mainitut luokat nidotaan yhteen Qt Creatorilla suunnitellun käyttöliittymän kanssa. Ylävalikosta valitaan laitteen AD-muuntimen asetukset: resoluutio, muunnosaika ja muuntimen kellotaajuus. Oikeassa laidassa on napit laitteeseen yhdistämiselle, tasajännitekomponentin kompensoinnille ja liipaisureunan valinnalle. Mittapään vaimennukselle on valintanapit 1x ja 10x. Oikeassa alakulmassa on kaksi säädintä, joilla valitaan oskilloskoopin aika- ja jänniteakseleiden skaalaus. Käyttöliittymässä olevilla tekstikentillä näytetään mittaustuloksia ja näytteenottotaajuus. Suurimman osan käyttöliittymästä vievät oskilloskoopin näyttö ja FFT:n näyttö, jotka ovat omilla välilehdillänsä.

Yhdistä-nappia painettaessa käsketään USB-käsittelijä yhdistämään laitteeseen sekä käynnistämään AD-muunnin.”Käytä maatasona”-nappi asettaa signaalin nykyisen jännitteen maatasoksi. Kun oskilloskoopin mittapää on kytketty maatasoon, jäljelle jää tulovahvistimen signaaliin lisäämä tasakomponentti. Tällä toiminnolla voidaan poistaa ylimääräinen tasajännite. Kolmannella napilla valitaan, käytetäänkö signaalin nousevaa vai laskevaa reunaa liipaisukohtana.

Vaimennuksen suuruus valitaan kahdella valintanapilla. Kun vaimennus on 10-kertainen, niin signaali kerrotaan mahdollisen tasajännitteen poiston jälkeen kymmenellä. Vaimennuksen ollessa yksi signaalille ei tehdä mitään. Oskilloskoopin näyttö on jaettu 10x10-kokoiseksi ruudukoksi. Käyttöliittymän kahdella säätimellä määritetään, kuinka montaa millivolttia ja mikrosekuntia jokainen ruutu vastaa. Kuvassa 7 on esitetty oskilloskooppi toiminnassa.



Kuva 7: Oskilloskooppi toiminnassa

Kun hiiren vasemmalla napilla valitaan oskilloskoopin näytöltä jokin piste ja sen jälkeen siirretään hiiren kohdistin johonkin toiseen kohtaan näyttöä, lasketaan pisteiden erotus. Erotuksen arvo näytetään käyttöliittymässä jännitteenä ja sekunteina. Ajasta lasketaan myös taajuus.

AD-muuntimen asetukset valitaan käyttöliittymän ylävalikosta. Valikossa on kolme alavalikkoa: resoluutio, kellotaajuus ja näytteistysaika. Alavalikoilla on alavalikoissaan mahdolliset arvot kyseiselle asetukselle. Esimerkiksi resoluution kohdalta aukeaa alavalikko, jossa on valinnat 6, 8, 10 ja 12. Kun jotain arvo klikataan, lähetetään USB-käsittelijän kautta laitteelle käsky muuttaa kyseistä arvoa. Arvojen muuttaminen muuttaa näytteenottotaajuutta, joten se päivitetään käskyn lähetyksen yhteydessä.

Pääohjelman suurin työ tehdään päivitys-funktiossa. Funktiota kutsutaan aina kun laitteelta on saatu uusi erä muunnostuloksia. Koska tulokset ovat kahdessa tavussa, mutta ne lähetetään USB:n yli yhden tavun paloissa, on arvot muutettava kaksitavuisiksi. Samalla arvoista vähennetään AD-tulovahvistimen lisäämä tasajännite ja skaalataan erotuksen jälkeen arvo vaimennuskertoimella. Sen jälkeen signaalista etsitään liipaisukoh-

ta. Mikäli nykyinen näyte on pienempi kuin seuraava näyte, todetaan että signaalissa on menossa nouseva reuna. Jos samaan aikaan näytteen arvo on suurempi kuin liipaisukohta, niin aloitetaan signaalin piirtäminen tästä kohdasta. Vastaavalla tavalla toimitaan jos liipaisureunaksi on valittu laskeva reuna.

Kun liipaisureuna on löydetty, tarkistetaan onko oskilloskooppi- vai FFT-välilehti aktiivisena. Jos oskilloskooppi on aktiivisena, kutsutaan oskilloskooppi-luokan päivitysfunktiota. Mikäli aktiivisena on FFT-välilehti, kutsutaan FFT-luokan päivitysfunktiota. Kun kuvaaja on piirretty näytölle, lähetetään laitteelle USB-käsittelijän kautta käsky käynnistää AD-muunnin. Näin tapahtumaketju alkaa alusta. Pääohjelman lähdekoodi on liitteessä 8.

6 YHTEENVETO

Kokonaisuutena työ onnistui kohtalaisen hyvin. AD-tulovahvistimen taajuus- ja askelvaste lienee ensimmäinen kehityskohde. Seuraavaksi ongelmaksi tulee näytteistysnopeus. Sitä voi jonkin verran parantaa pelkällä ohjelmistomuutoksella, mutta seuraava järkevä kehitysaskel lienee erillisen AD-muuntimen käyttö.

Koska mikrokontrollerilla vain muunnetaan ja lähetetään signaali tietokoneelle, on ominaisuuksien lisääminen kohtalaisen helppoa. Ohjelmallisesti voidaan toteuttaa esimerkiksi pidempiaikaista näytteiden keruuta, erilaisia suodattimia tai yleisesti ottaen mitä tahansa signaalinkäsittelyä. Normaalissa tietokoneessa on riittävästi laskentatehoa reaaliaikaiseen signaalinkäsittelyyn. Ilman suurempia elektronisia muutoksia, olisi mahdollista lisätä kaksi rinnakkaista mittauskanavaa.

Erillisellä AD-muuntimella päästäisiin satojen megahertsien näytteenottotaajuuksiin. Nopean AD-muuntimen pariksi tarvittaisiin nopea mikrokontrolleri hoitamaan tiedonsiirto. Todennäköisesti FPGA-piiri olisi vaaditussa nopeusluokassa halvempi vaihtoehto mikrokontrollerille. Myös USB-yhteys on samalla päivitettävä nopeampaan standardiin. Tiedon suuri määrä voi mahdollisesti vaatia tietokoneeltakin jo hieman enemmän laskentatehoa, varsinkin jos tehdään raskaampaa signaalinkäsittelyä.

LÄHTEET

1 USB in a NutShell
[online][viitattu 8.5.2012]
http://www.beyondlogic.org/usbnutshell/usb1.shtml

2 Communication in the Presence of Noise
[online][viitattu 8.5.2012]
http://www.stanford.edu/class/ee104/shannonpaper.pdf

3 Signal to quantization noise in quantized sinusoidal
[online][viitattu 8.5.2012]
http://www.dsplog.com/2007/03/19/signal-to-quantization-noise-in-quantized-sinusoidal/

4 STM32F4-DISCOVERY
[online][viitattu 8.5.2012]
http://www.st.com/internet/evalboard/product/252419.jsp

5 Developing USB applications using the STM32 ARM Cortex-M3 micro controller
[online][viitattu 8.5.2012]
www.st.com/internet/com/CORPORATE_RESOURCES/COMPANY/DIVISIONAL_PRESENTATION/Nuremberg_Workshop_USB.pdf

6 libusb-win32
[online][viitattu 8.5.2012]
http://sourceforge.net/apps/trac/libusb-win32/wiki/

7 libusb-win32 -kirjasto
[online][viitattu 8.5.2012]
http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/1.2.6.0/

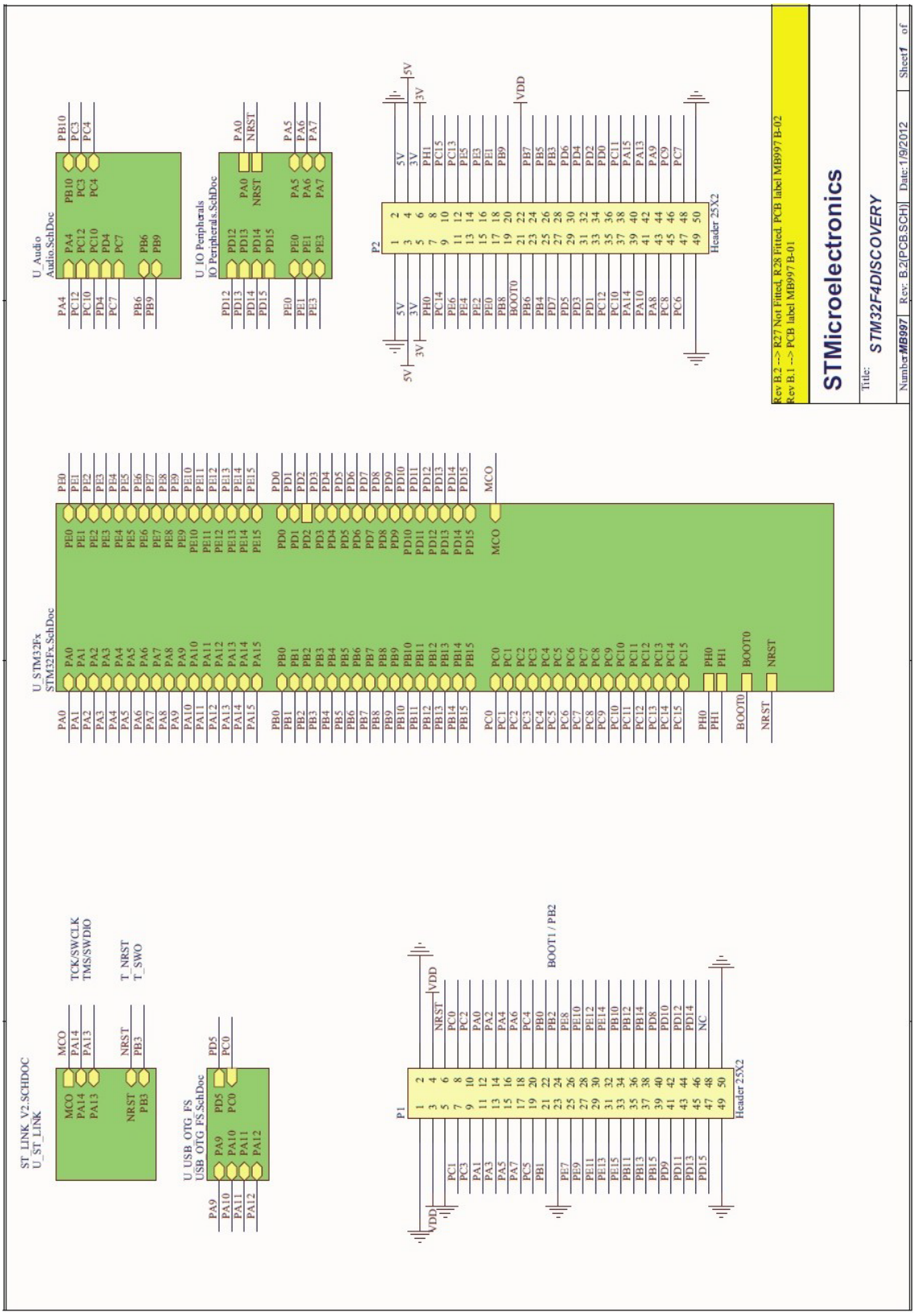
8 Fast Fourier transform – FFT – Librow – Software
[online][viitattu 8.5.2012]
http://www.librow.com/articles/article-10

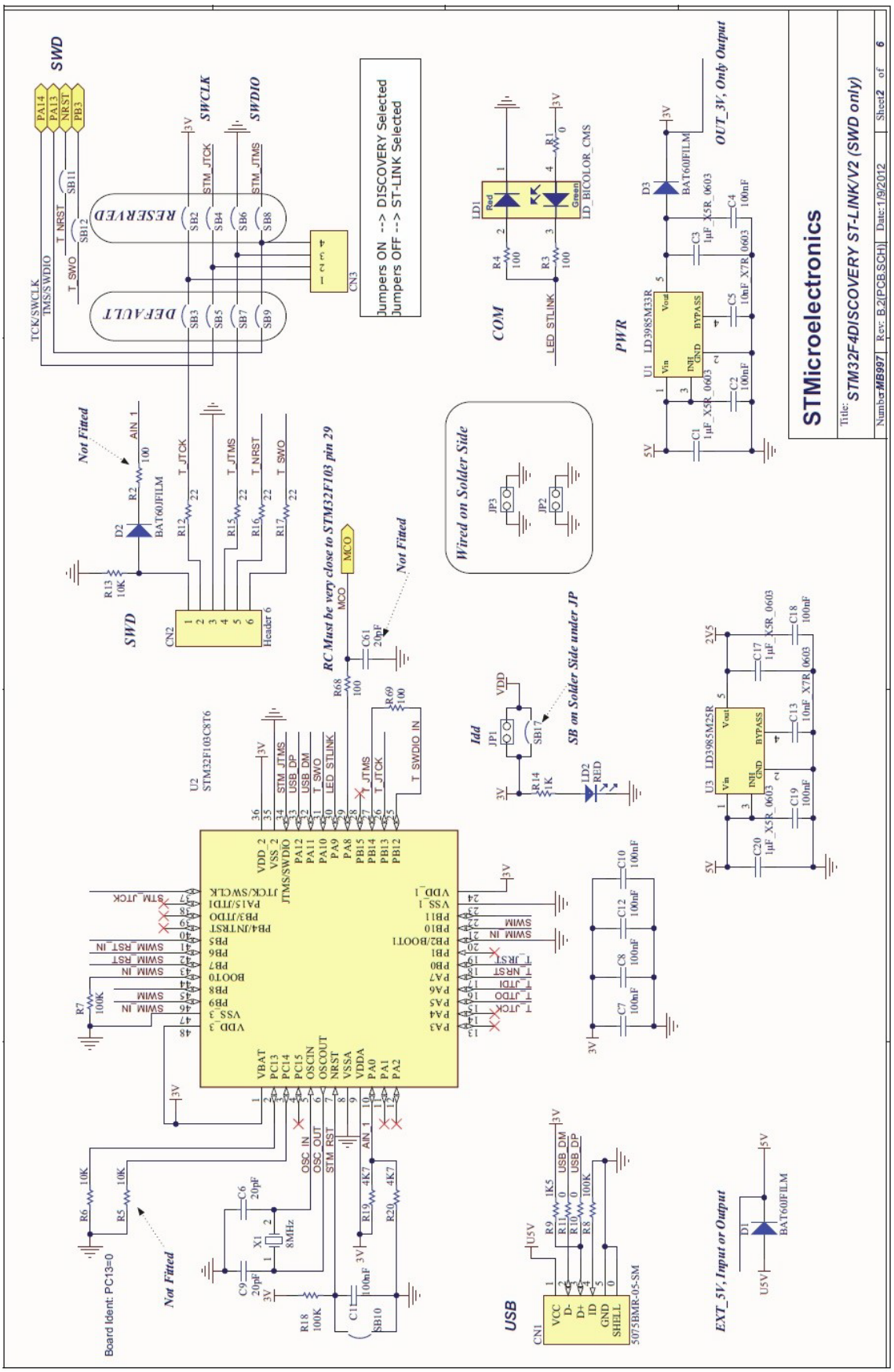
9 FFT — C++ source code
[online][viitattu 8.5.2012]
http://www.librow.com/content/en/download/articles/article-10/fft_code.zip

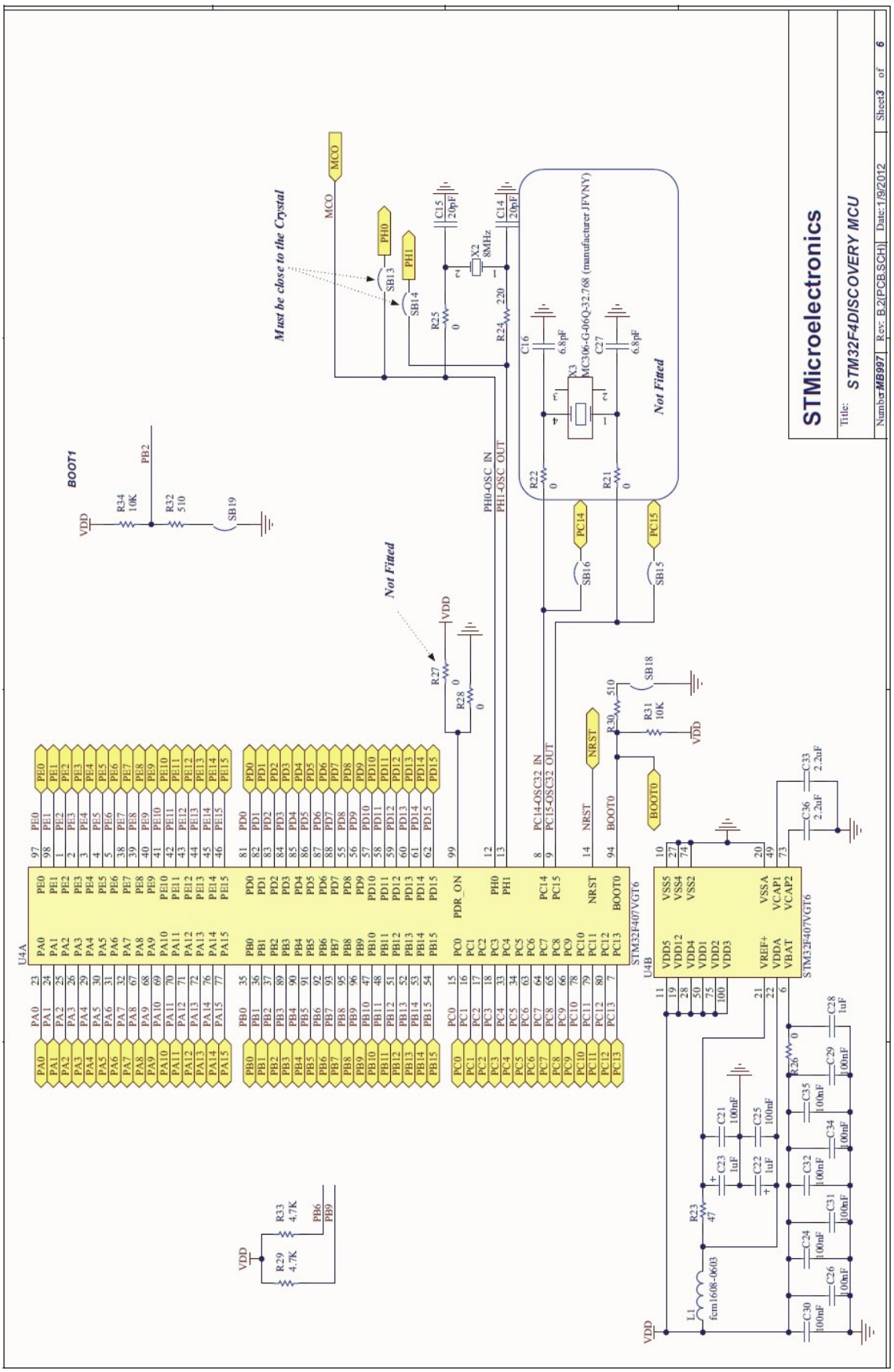
10 STM32F4 high-performance discovery board
[online][viitattu 8.5.2012]
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/USER_MANUAL/DM00039084.pdf

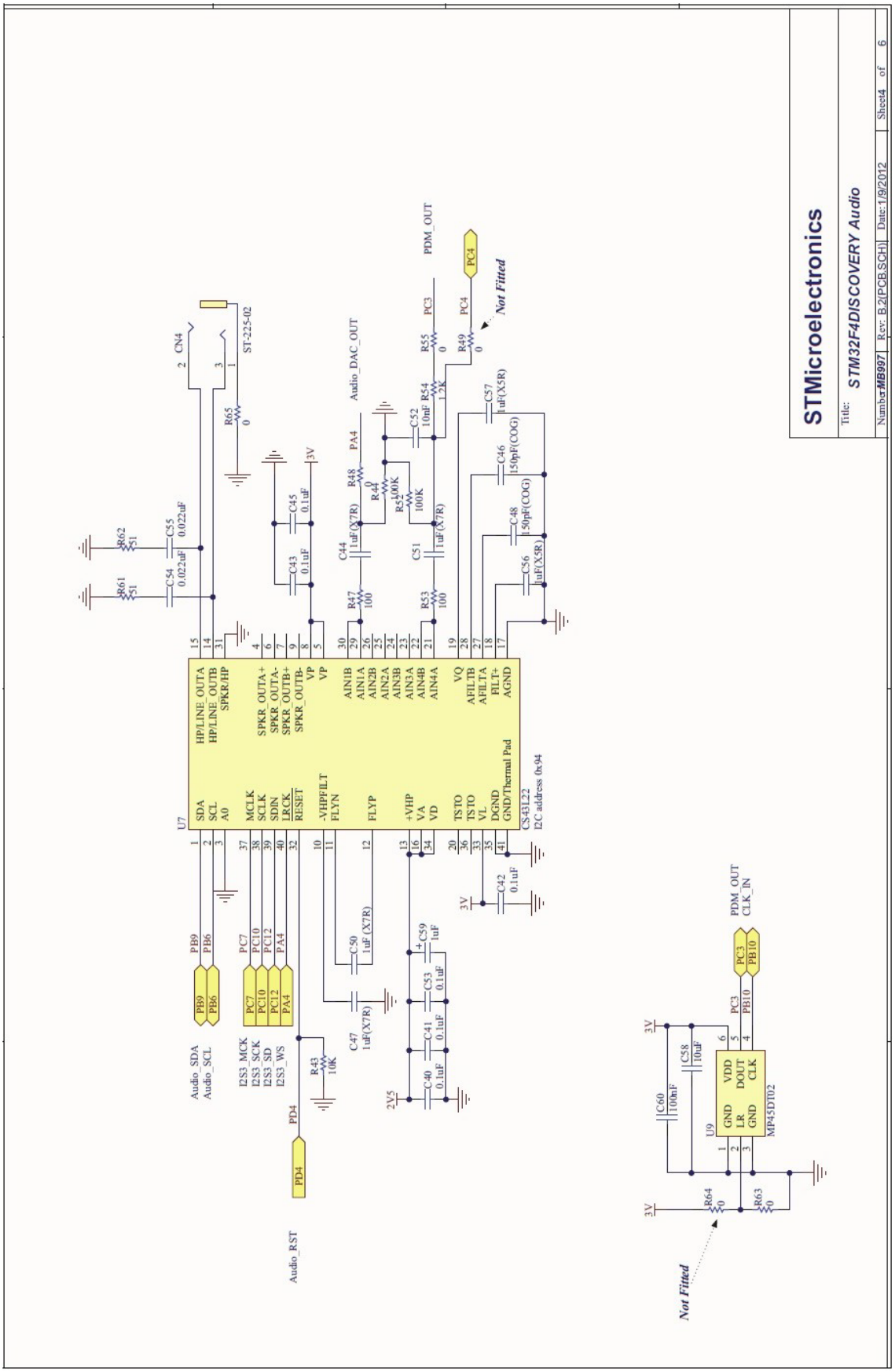
LIITTEET

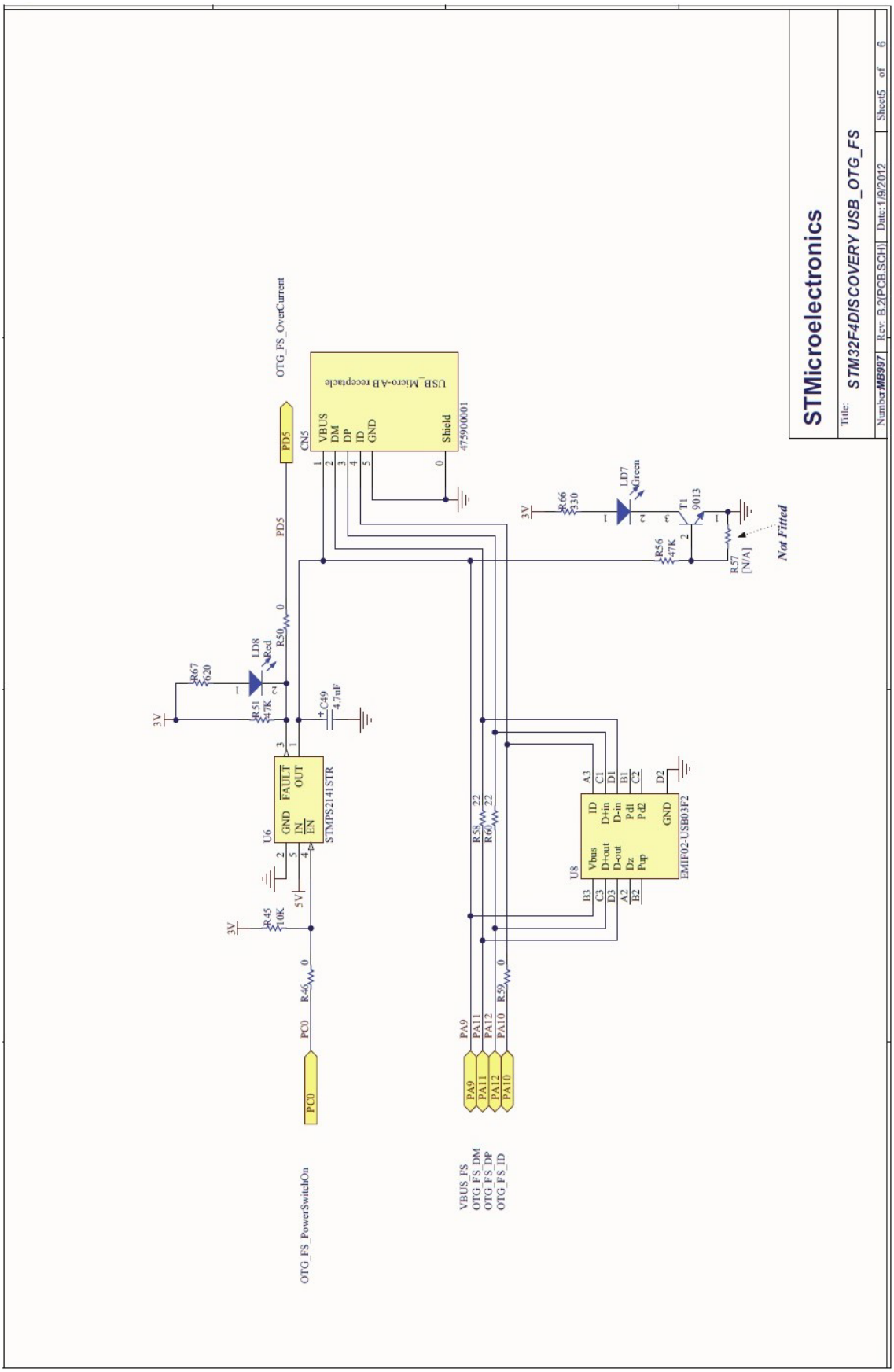
Liite 1. Käytetyn kehitysalustan kytkentäkaavio [10].

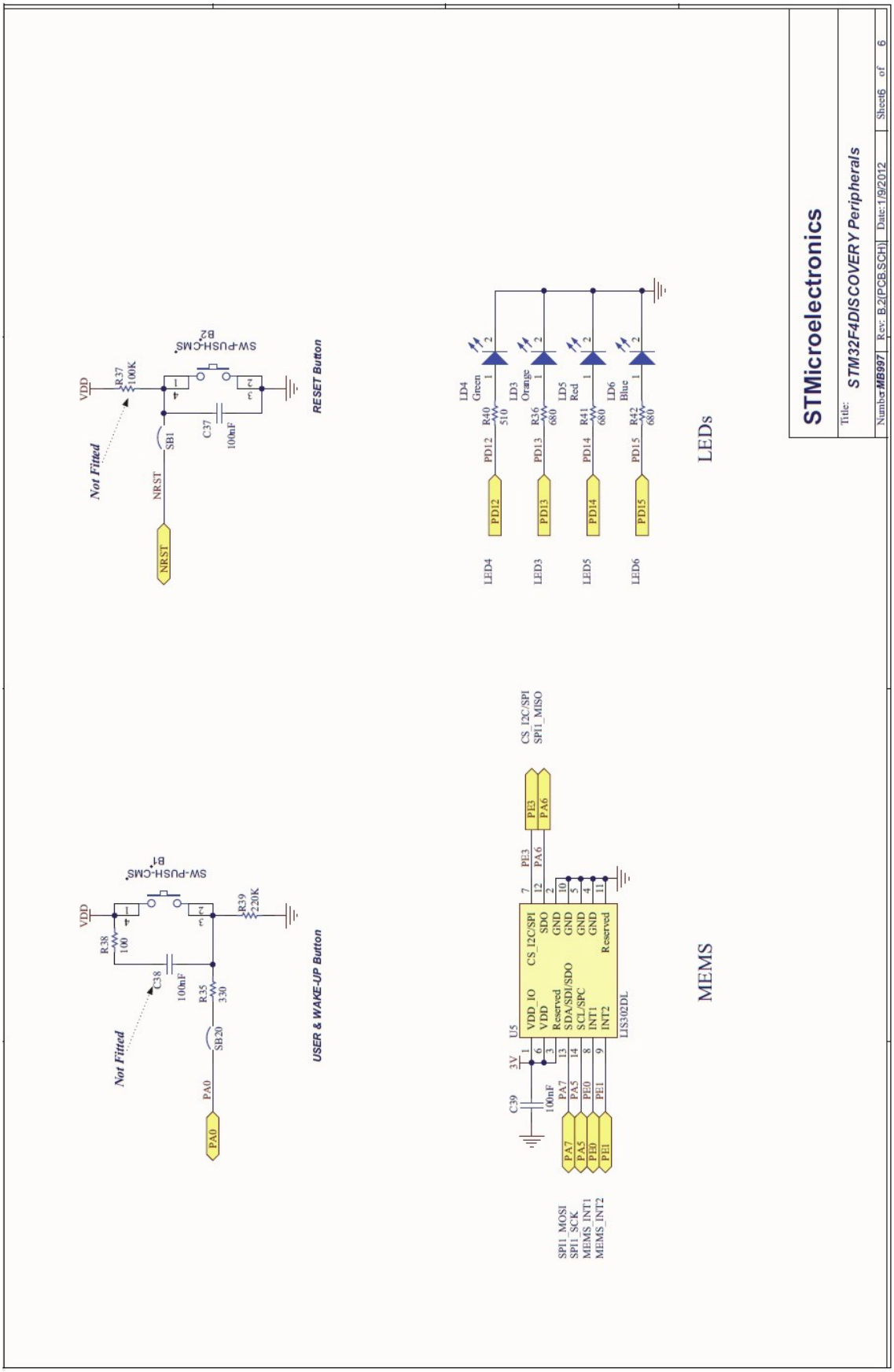



STMicroelectronics
Title: STM32F4DISCOVERY ST-LINK/V2 (SWD only)
Number: MB997 Rev: B.2(PCB.SCH) Date: 1/9/2012 Sheet 2 of 6
SWD
SWCLK
SWDIO
RESERVED
DEFAULT
Jumpers ON --> DISCOVERY Selected
Jumpers OFF --> ST-LINK Selected
COM
PWR
OUT_3V, Only Output
Wired on Solder Side
Not Fitted
RC Must be very close to STM32F103 pin 29
SB on Solder Side under JP
Idd
USB
EXT_5V, Input or Output
Board Ident: PC13=0
STM32F103C8T6
LD3985M33R
LD3985M25R
BAT60JFILM
LD_BICOLOR_CMS
5073BMR-05-SM


STMicroelectronics
Title: STM32F4DISCOVERY MCU
Number: MB997
Rev: B.2(PCB.SCH)
Date: 1/9/2012
Sheet 3 of 6
Must be close to the Crystal
Not Fitted
Not Fitted
BOOT1
MCO
PH0
PH1
PH0-OSC_IN
PH1-OSC_OUT
PC14-OSC32_IN
PC15-OSC32_OUT
NRST
BOOT0
PDR_ON
X2 8MHz
X3 MC306-G-06Q-32.768 (manufacturer JFVNY)
U4A
U4B
STM32F407VGT6


STMicroelectronics
Title: STM32F4DISCOVERY Audio
Number: MB997 Rev: B.2[PCB.SCH] Date: 1/9/2012 Sheet4 of 6
U7
CS43122
I2C address 0x94
HP/LINE_OUTA
HP/LINE_OUTB
SPKR/HP
SPKR_OUTA+
SPKR_OUTA-
SPKR_OUTB+
SPKR_OUTB-
SDA
SCL
A0
MCLK
SCLK
SDIN
LRCK
RESET
-VHPFILT
FLYN
FLYP
+VHP
VA
VD
TSTO
TSTO
VL
DGND
GND/Thermal Pad
VP
AIN1B
AIN1A
AIN2B
AIN2A
AIN3B
AIN3A
AIN4B
AIN4A
VQ
AFILTB
AFILTA
FILT+
AGND
CN4
ST-225-02
Audio_DAC_OUT
PDM_OUT
Not Fitted
Audio_SDA
Audio_SCL
I2S3_MCK
I2S3_SCK
I2S3_SD
I2S3_WS
Audio_RST
U9
MP45DT02
GND
LR
GND
VDD
DOUT
CLK
PDM_OUT
CLK_IN
Not Fitted


STMicroelectronics
Title: STM32F4DISCOVERY USB_OTG_FS
Number:MB997 Rev: B.2(PCB.SCH) Date:1/9/2012 Sheet5 of 6
USB_Micro-AB receptacle
CN5
VBUS
DM
DP
ID
GND
Shield
47590001
OTG_FS_OverCurrent
PD5
LD7 Green
R66 330
T1 9013
R56 47K
R57 [N/A]
Not Fitted
LD8 Red
R67 620
R51 47K
R50 0
C49 4.7uF
U6
STMPS2141STR
GND
FAULT
OUT
IN
EN
R45 10K
R46 0
PC0
OTG_FS_PowerSwitchOn
U8
EMIF02-USB03F2
Vbus
D+out
D-out
Dz
Pup
ID
D+in
D-in
Pd1
Pd2
GND
R58 22
R60 22
R59 0
PA9
PA11
PA12
PA10
VBUS_FS
OTG_FS_DM
OTG_FS_DP
OTG_FS_ID
3V
5V


STMicroelectronics
Title: STM32F4DISCOVERY Peripherals
Number:MB997 Rev: B.2(PCB.SCH) Date:1/9/2012 Sheet6 of 6
RESET Button
SW-PUSH-CMS
B2
VDD
R37
100K
Not Fitted
SB1
NRST
C37
100nF
LEDs
LD4
Green
LD3
Orange
LD5
Red
LD6
Blue
R40
510
R36
680
R41
680
R42
680
PD12
PD13
PD14
PD15
LED4
LED3
LED5
LED6
USER & WAKE-UP Button
SW-PUSH-CMS
B1
R38
100
C38
100nF
R39
220K
R35
330
SB20
PA0
MEMS
U5
LIS302DL
VDD_IO
VDD
Reserved
SDA/SDI/SDO
SCL/SPC
INT1
INT2
CS_I2C/SPI
SDO
GND
Reserved
C39
100nF
3V
PA7
PA5
PE0
PE1
PE3
PA6
CS_I2C/SPI
SPI1_MISO
SPI1_MOSI
SPI1_SCK
MEMS_INT1
MEMS_INT2

Liite 2. Mikrokontrollerin pääohjelman lähdekoodi.

Main.c

```
#include "stm32f4xx.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_dma.h"
#include "usbd_cdc_core.h"
#include "usbd_cdc.h"
#include "usbd_usr.h"
#include "usbd_desc.h"
//muunnostuloksen osoite
#define ADC1_DR_ADDRESS ((uint32_t) 0x4001204C )
//muunnostulosten puskuri
uint16_t adcMuunnokset[APP_RX_DATA_SIZE/2];
//ad - muuntimen asetukset
char preScaler, sampleTime, sampleDelay, resolution;
//funktioiden prototyypit
void Delay(__IO uint32_t nTick);
void DMAconfig( void );
void ADCconfig( char preScaler, char sampleTime,  char resolution );

__ALIGN_BEGIN USB_OTG_CORE_HANDLE    USB_OTG_dev __ALIGN_END ;

int main(void)
{
	Delay(0xFFFF);
	//ad-muuntimen asetukset.
	preScaler = 2;
	sampleTime = 3;
	resolution = 12;

	//usb alustus
	USBD_Init(&USB_OTG_dev, USB_OTG_FS_CORE_ID, &USR_desc, &USBD_CDC_cb, &USR_cb);

	//laitteiden kellot käyntiin
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
	RCC_AHB1PeriphClockCmd( RCC_AHB1Periph_DMA2, ENABLE );
	RCC_APB2PeriphClockCmd( RCC_APB2Periph_ADC1, ENABLE );

	//portti C:n pinni 2 analogiseksi.
	GPIO_InitTypeDef GPIO_InitStructure;
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init( GPIOC, &GPIO_InitStructure );

	//asetukset
	DMAconfig();
	ADCconfig( preScaler, sampleTime, sampleDelay, resolution );

	//toiminta tapahtuu keskeytyspalvelussa
	//joten pääsilmukassa ei tehdä mitään
	while (1)
	{
	}
}

void Delay(__IO uint32_t nTick)
{
	for(; nTick != 0; nTick--);
}
```

```
void DMAconfig( void )
{
	DMA_InitTypeDef DMA_IS;
	NVIC_InitTypeDef NVIC_IS;

	NVIC_IS.NVIC_IRQChannel = DMA2_Stream0_IRQn;
	NVIC_IS.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_IS.NVIC_IRQChannelSubPriority = 0;
	NVIC_IS.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_IS);

	DMA_DeInit(DMA2_Stream0);
	DMA_IS.DMA_Channel = DMA_Channel_0;
	DMA_IS.DMA_PeripheralBaseAddr = (uint32_t)ADC1_DR_ADDRESS;
	DMA_IS.DMA_Memory0BaseAddr = (uint32_t)&adcMuunnokset;
	DMA_IS.DMA_DIR = DMA_DIR_PeripheralToMemory;
	DMA_IS.DMA_BufferSize = APP_RX_DATA_SIZE/2;
	DMA_IS.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
	DMA_IS.DMA_MemoryInc = DMA_MemoryInc_Enable;
	DMA_IS.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
	DMA_IS.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
	DMA_IS.DMA_Mode = DMA_Mode_Circular;
	DMA_IS.DMA_Priority = DMA_Priority_High;
	DMA_IS.DMA_FIFOMode = DMA_FIFOMode_Disable;
	DMA_IS.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
	DMA_IS.DMA_MemoryBurst = DMA_MemoryBurst_Single;
	DMA_IS.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
	DMA_Init(DMA2_Stream0, &DMA_IS);
	DMA_ITConfig( DMA2_Stream0, DMA_IT_TC, ENABLE );
	DMA_Cmd(DMA2_Stream0, ENABLE);
}

void ADCconfig( char preScaler, char sampleTime, char resolution )
{

	ADC_Cmd( ADC1, DISABLE );
	ADC_DeInit();
	uint32_t ps, r;
	uint8_t st;
	//Säädetään prescalerin arvo structin ymmärtämään muotoon
	switch( preScaler )
	{
	case 2:
		ps = ADC_Prescaler_Div2;
		break;
	case 4:
		ps = ADC_Prescaler_Div4;
		break;
	case 6:
		ps = ADC_Prescaler_Div6;
		break;
	case 8:
		ps = ADC_Prescaler_Div8;
		break;
	default:
		ps = ADC_Prescaler_Div2;
		break;
	}
	//säädetään smple time arvo sopivaksi
	switch( sampleTime )
	{
	case 3:
		st = ADC_SampleTime_3Cycles;
		break;
	case 15:
		st = ADC_SampleTime_15Cycles;
		break;
	case 28:
		st = ADC_SampleTime_28Cycles;
		break;
```

```
		case 56:
			st = ADC_SampleTime_56Cycles;
			break;
		case 84:
			st = ADC_SampleTime_84Cycles;
			break;
		case 112:
			st = ADC_SampleTime_112Cycles;
			break;
		case 144:
			st = ADC_SampleTime_144Cycles;
			break;
		default:
			st = ADC_SampleTime_480Cycles;
			break;
		}
		//säädetään resoluutio oikeaan muotoon
		switch( resolution )
		{
		case 12:
			r = ADC_Resolution_12b;
			break;
		case 10:
			r = ADC_Resolution_10b;
			break;
		case 8:
			r = ADC_Resolution_8b;
			break;
		case 6:
			r = ADC_Resolution_6b;
			break;
		default:
			r = ADC_Resolution_8b;
			break;
		}

		ADC_InitTypeDef ADC_IS;
		ADC_CommonInitTypeDef ADC_CIS;

		ADC_CIS.ADC_Mode = ADC_Mode_Independent;
		ADC_CIS.ADC_Prescaler = ps;
		ADC_CIS.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
		ADC_CIS.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
		ADC_CommonInit(&ADC_CIS);

		ADC_IS.ADC_Resolution = r;
		ADC_IS.ADC_ScanConvMode = DISABLE;
		ADC_IS.ADC_ContinuousConvMode = ENABLE;
		ADC_IS.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
		ADC_IS.ADC_DataAlign = ADC_DataAlign_Right;
		ADC_IS.ADC_NbrOfConversion = 1;
		ADC_Init(ADC1, &ADC_IS);

		ADC_DMACmd(ADC1, ENABLE);
		ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 1, st);
		ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);
		ADC_Cmd(ADC1, ENABLE);
}
```

Liite 3. Mikrokontrollerin keskeytyspalvelun lähdekoodi.

stm32f4xx_it.h

```
#ifndef __STM32F4xx_IT_H
#define __STM32F4xx_IT_H

#include "stm32f4xx.h"

void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);
void DMA2_Stream0_IRQHandler(void);
void muutaVolteiksi( void );

#endif
```

stm32f4xx_it.c

```
#include "stm32f4xx_it.h"
#include "usb_core.h"
#include "usbd_core.h"
#include "stm32f4_discovery.h"
#include "usbd_cdc_core.h"

extern uint32_t APP_Rx_ptr_in;
extern uint8_t  APP_Rx_Buffer [];

extern void dataTx( void );

extern USB_OTG_CORE_HANDLE          USB_OTG_dev;
extern uint32_t USBD_OTG_ISR_Handler (USB_OTG_CORE_HANDLE *pdev);

extern uint16_t adcMuunnokset[];

void NMI_Handler(void)
{
}

void HardFault_Handler(void)
{
        while (1)
        {
        }
}

void MemManage_Handler(void)
{
        while (1)
        {
        }
}

void BusFault_Handler(void)
{
        while (1)
        {
```

```
	}
}

void UsageFault_Handler(void)
{
	while (1)
	{
	}
}

void SVC_Handler(void)
{
}

void DebugMon_Handler(void)
{
}

void PendSV_Handler(void)
{
}

void SysTick_Handler(void)
{
}

void OTG_FS_WKUP_IRQHandler(void)
{
	if(USB_OTG_dev.cfg.low_power)
	{
		*(uint32_t *)(0xE000ED10) &= 0xFFFFFFF9 ;
		SystemInit();
		USB_OTG_UngateClock(&USB_OTG_dev);
	}
	EXTI_ClearITPendingBit(EXTI_Line18);
}

void OTG_FS_IRQHandler(void)
{
	USBD_OTG_ISR_Handler (&USB_OTG_dev);
}

void DMA2_Stream0_IRQHandler(void)
{
	if(DMA_GetITStatus(DMA2_Stream0, DMA_IT_TCIF0))
	{
		//ad-muunnin kiinni
		ADC_Cmd( ADC1, DISABLE );
		//nollataan keskeytyslippu
		DMA_ClearITPendingBit(DMA2_Stream0, DMA_IT_TCIF0);
		//muutetaan arvot volteiksi
		muutaVolteiksi();
		int e = 0, i = 0;
		//kopioidaan voltit usb - puskuriin
		for(i = 0; i < APP_RX_DATA_SIZE/2; i++ )
		{
			//alemmat 8 bittiä
			APP_Rx_Buffer[e+1] = adcMuunnokset[i] & 0xFF;
			//ylemmät 8 bittiä
			APP_Rx_Buffer[e] = adcMuunnokset[i] >> 8;
			e += 2;
		}
		//osoitin puskurin loppuun
		APP_Rx_ptr_in = APP_RX_DATA_SIZE;
		//ilmoitus usb-ajureille
		dataTx(  );
	}
}
```

```
void muutaVolteiksi( void )
{
	uint16_t kerroin = 0;
	if( resolution == 6 )
		kerroin = 64;
	else if( resolution == 8 )
		kerroin = 256;
	else if( resolution == 10 )
		kerroin = 1024;
	else
		kerroin = 4096;

	int i;
	for( i = 0; i < APP_RX_DATA_SIZE/2; i++ )
	{
		adcMuunnokset[i] = adcMuunnokset[i]*3300/kerroin;
	}
}
```

Liite 4. Mikrokontrollerin USB-lähetyksen ja vastaanoton lähdekoodi.

usbd_cdc.c

```
#include "usbd_cdc.h"

extern void ADCconfig( char preScaler, char sampleTime, char resolution );
extern void DMAconfig( void );

extern uint8_t  APP_Rx_Buffer [];
extern uint32_t APP_Rx_ptr_in;
extern char preScaler, sampleTime, resolution;

static uint16_t cdc_Init     (void);
static uint16_t cdc_DeInit   (void);
static uint16_t cdc_Ctrl     (uint32_t Cmd, uint8_t* Buf, uint32_t Len);
static uint16_t cdc_DataTx   (uint8_t* Buf, uint32_t Len);
static uint16_t cdc_DataRx   (uint8_t* Buf, uint32_t Len);

CDC_IF_Prop_TypeDef cdc_fops =
{
  cdc_Init,
  cdc_DeInit,
  cdc_Ctrl,
  cdc_DataTx,
  cdc_DataRx
};

static uint16_t cdc_Init(void)
{
	return USBD_OK;
}

static uint16_t cdc_DeInit(void)
{
	return USBD_OK;
}

static uint16_t cdc_Ctrl (uint32_t Cmd, uint8_t* Buf, uint32_t Len)
{
	return USBD_OK;
}

//datan lähetys
//toimii signaalina usb - ajureille että puskuriin on kirjoitettu
//lähetettävää tietoa.
static uint16_t cdc_DataTx (uint8_t* Buf, uint32_t Len)
{
	return USBD_OK;
}
//datan vastaanotto
//usb - ajurit kutsuvat funktiota kun dataa on vastaanotettu.
static uint16_t cdc_DataRx (uint8_t* Buf, uint32_t Len)
{
	uint32_t i;
	//käydään puskuri läpi
	for (i = 0; i < Len; i++)
	{
		//jos löytyi 'b'
		if( *(Buf + i) == 'b' )
		{
			//laitetaan ad - muunnin päälle
			ADC_Cmd( ADC1, ENABLE );
			ADC_SoftwareStartConv(ADC1);
		}
		//jos löytyi 'D'
```

```
                if( *(Buf + i ) == 'D' && Len - i > 1 && i < 3 )
                {
                        //vaihdetaan esijakaja ja ajetaan asetukset uudelleen
                        preScaler = Buf[i+1];
                        DMAconfig();
                        ADCconfig(preScaler,sampleTime,sampleDelay,resolution );
                        ADC_SoftwareStartConv(ADC1);
                }
                //jos löytyi 'T'
                if( *(Buf + i ) == 'T' && Len - i > 1 && i < 3 )
                {
                        //vaihdetaan näytteistysaika ja ajetaan asetukset uudelleen
                        sampleTime = Buf[i + 1];
                        DMAconfig();
                        ADCconfig(preScaler,sampleTime,sampleDelay,resolution );
                        ADC_SoftwareStartConv(ADC1);
                }
                //jos löytyi 'W'
                if( *(Buf + i ) == 'W' && Len - i > 1 && i < 3 )
                {
                        //vaihdetaan muunnostarkkuus
                        resolution = Buf[i + 1];
                        DMAconfig();
                        ADCconfig(preScaler,sampleTime,sampleDelay,resolution );
                        ADC_SoftwareStartConv(ADC1);
                }
        return USBD_OK;
}

//koska lähetysfunktio on esitelty static määreellä
//niin esitellään funktio jota voi kutsua myös muista ohjelman osioista.
void dataTx( void  )
{
        cdc_DataTx( "as", 2 );
}
```

Liite 5. Pääohjelman USB-käsittelijän lähdekoodi.

usbHandler.h

```
#ifndef USBHANDLER_H
#define USBHANDLER_H
#include "../libusb-win32-bin-1.2.6.0/include/lusb0_usb.h"
#include <QObject>
#include <QString>
#include <Qtimer>

class usbHandler : public QObject
{
	Q_OBJECT

public:
	usbHandler( QObject *parent = 0 );
	~usbHandler();
	void connectToDevice();
	void sendCommand( char *command );
	void getData( char *dst );
	QByteArray buffer;

signals:
	void readReady();

private:
	QTimer *timer;
	char readBuffer[8192];
	int byteReadCount;
	int pid;
	int vid;
	int conf;
	int intf;
	int ep_in;
	int ep_out;
	usb_dev_handle *device;

private slots:
	void readData();
};

#endif // USBHANDLER_H
```

usbHandler.cpp

```
#include <usbHandler.h>
//rakentaja, alustetaan usb ajurit
usbHandler::usbHandler( QObject *parent )
{
	usb_init();
	pid = 0xe458;
	vid = 0x305;
	conf = 1;
	intf = 1;
	ep_in = 0x81;
	ep_out = 0x01;

	device = NULL;
	//pistetään ajastin päälle jolla tahdistetaan data lukemista
	timer = new QTimer( this );
	//joka kerta kun "herätyskello" soi niin kutsutaan funktiota readData
	connect(timer, SIGNAL( timeout() ), this, SLOT( readData() ) );
}
```

```
//kutsutaan kun luokka tuhotaan, eli käytännössä ohjelman lopussa.
usbHandler::~usbHandler()
{
	//pysäytetään kello
	timer->stop();
	delete timer;
	//vapautetaan liittymä
	usb_release_interface( device, intf );
	//suljetaan laite
	usb_close( device );
}

//kutsutaan kun käyttöliittymästä painetaan "yhdistä" nappia.
void usbHandler::connectToDevice()
{
	//sisältää koneesta löytyvät usb-linjat
	struct usb_bus *bus;
	//sisältää koneesta löytyvät usb-laitteet
	struct usb_device *dev;
	qDebug("yhdistetään");
	//etsitään koneesta löytyvät usb-linjat
	usb_find_busses();
	//etsitään koneeseen kytketyt usb-laitteet
	usb_find_devices();
	//muuttuja jonka avulla hypätään ensimmäisen löytyvän laitteen yli
	bool first = true;
	//käydään kaikki löytyneet laitteet läpi
	for (bus = usb_get_busses(); bus; bus = bus->next)
	{
		for (dev = bus->devices; dev; dev = dev->next)
		{
			//jos vendor ID ja product ID ovat oikeat
			if (dev->descriptor.idVendor == vid && dev->descriptor.idProduct == pid)
			{
			//hypätään ensimmäisen löytyneen laitteen yli koska se on väärä
				if( first )
					first = false;
				//toinen laite jonka VID ja PID täsmäävät on oikea laite.
				else
					//avataan laite;
					device = usb_open( dev );
			}
		}
	}

	//jos saatiin laite auki niin tehdään loputkin asetukset
	if( device != NULL)
	{
		if( usb_set_configuration( device, conf ) < 0 )
			qDebug( usb_strerror() );
		//napataan laitteen rajapinta käyttöön
		if( usb_claim_interface( device, intf ) < 0 )
			qDebug( usb_strerror() );
		char message[64];
		message[0] = 'b';
		usb_bulk_write(device, ep_out, message, 64, 100 );
		//kutsutaan lukufunktiota
		timer->setSingleShot(true);
		timer->start( 1 );
	}
	else
		qDebug( usb_strerror() );
}
```

```
//lähettää laitteelle viestin
void usbHandler::sendCommand( char* command )
{
	usb_bulk_write(device, ep_out, command, sizeof(command), 100 );
	if( command[0] == 'b' )
	{
		timer->setSingleShot(true);
		timer->start( 1 );
	}
}

void usbHandler::readData()
{
	byteReadCount = usb_bulk_read(device, ep_in, readBuffer, 8192, 100 );
	emit readReady();
}

void usbHandler::getData( char *dst )
{
	 memcpy(dst, readBuffer, sizeof(readBuffer));
}
```

Liite 6. Pääohjelman oskilloskooppi-luokan lähdekoodi.

oscScene.h

```
#ifndef OSCSCENE_H
#define OSCSCENE_H
#include <QObject>
#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
class oscScene : public QGraphicsScene
{
	Q_OBJECT
public:
	oscScene( QObject *parent = 0 );
	~oscScene();
	void myUpdate( float *data,  int len,  int trigPos,  int voltsPerPixel, float timePerPixel );
signals:
	void trigPosChanged( qreal jannite );
	void measPointChanged( qreal v, qreal t );

protected:
	void mousePressEvent( QGraphicsSceneMouseEvent *mouseEvent );
	void mouseMoveEvent(QGraphicsSceneMouseEvent *event);

private:
	QPointF measPointA;
	QPointF measPointB;
	QPainterPath graph;
	QPainterPath grid;
	QPen lineStyle;
};
#endif // OSCSCENE_H
```

oscScene.cpp

```
#include "oscScene.h"
oscScene::oscScene( QObject *parent ) : QGraphicsScene( parent )
{
	measPointA.setX(0);
	measPointA.setY(0);
	measPointB.setX(0);
	measPointB.setX(0);
	//luodaan ruudukko skoopin taustalle
	for( int i = 60; i < 600; i += 60 )
	{
		//viiva pisteestä (i,0) pisteeseen (i, max korkeus)
		grid.moveTo( i, 0 );
		grid.lineTo( i, 500 );
	}
	//piirrellään 10 vaakaviivaa skoopin näyttöön
	for( int i = 50; i < 500; i += 50 )
	{
		//viiva pisteestä (0, i) pisteeseen (max leveys, i)
		grid.moveTo( 0, i );
		grid.lineTo( 600, i);
	}
	//viivan piirtotyyli
	lineStyle = QPen(Qt::NoBrush, 1, Qt::DashLine, Qt::SquareCap, Qt::BevelJoin );
	lineStyle.setColor(Qt::lightGray);
}

oscScene::~oscScene()
{
}
```

```
void oscScene::myUpdate(float *data, int len, int trigPos, int voltsPerPixel,
                                        float timePerPixel )
{
        //lähtökohta nollaksi
        qreal x = 0;
        //ruutu tyhjäksi
        this->clear();
        //taustaruudukko näytölle
        this->addPath( grid, lineStyle, Qt::NoBrush );
        //tyhjätään edellinen kuvaaja
        graph = QPainterPath();
        //aloitetaan kuvaajan piirtäminen ensimmäisen tuloksen kohdasta
        graph.moveTo(x, 500 - (data[trigPos] / voltsPerPixel ) );
        //rakennetaan kuvaaja muunnostuloksista
        for( int i = trigPos; i < len; i++ )
        {
                //piirretään edellisestä pisteestä viiva seuraavaan
                graph.lineTo(x, 500 - (data[i] / voltsPerPixel ) );
                // kasvatetaan piirtokohdan x-akselin arvoa
                x += timePerPixel;
                //jos arvo meni yli näytön reunan
                if( x > 600 )
                //poistutaan silmukasta
                break;
        }
        //lisätään kuvaaja näytölle
        this->addPath( graph, QPen( Qt::black), QBrush( Qt::NoBrush ) );
}

void oscScene::mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent)
{
        if( mouseEvent->button() == Qt::LeftButton )
        {
                measPointA = mouseEvent->scenePos();
        }
        if( mouseEvent->button() == Qt::RightButton )
        {
                QPointF piste = mouseEvent->scenePos();
                emit( trigPosChanged( piste.y() ) );
        }
}

void oscScene::mouseMoveEvent( QGraphicsSceneMouseEvent *event )
{
        measPointB = event->scenePos();
        qreal v, t;
        t = measPointB.x() - measPointA.x();
        v = measPointA.y() - measPointB.y();
        emit( measPointChanged( v, t ) );
}
```

Liite 7. Pääohjelman FFT-luokan lähdekoodi.

fftScene.h

```
#ifndef FFTKUVAAJA_H
#define FFTKUVAAJA_H
#include <QObject>
#include <QGraphicsScene>
#include <QgraphicsSceneMouseEvent>
//lähde: http://www.librow.com/articles/article-10
#include "FFT_CODE/fft.h"
class fftScene : public QGraphicsScene
{
	Q_OBJECT

public:
	fftScene( QObject *parent = 0 );
	~fftScene();
	void myUpdate( float *data, int len );
	void myClear();

private:
	complex *pData;
	double maxValues[4096];
	QPainterPath maxGraph;
	QPainterPath graph;
};
#endif // FFTKUVAAJA_H
```

fftScene.cpp

```
#include "fftScene.h"
fftScene::fftScene( QObject *parent )
{
	pData = new complex[4096];
}

fftScene::~fftScene()
{
	delete [] pData;
	pData = NULL;
}

void fftScene::myClear()
{
	this->clear();
	graph = QPainterPath();
	maxGraph = QPainterPath();
	for( int i = 0; i < 2048; i++ )
		maxValues[i] = 0;
}
void fftScene::myUpdate(float *data, int len)
{
	this->clear();
	graph = QPainterPath();
	maxGraph = QPainterPath();
	for( int i = 0; i < len; i++ )
	{
		pData[i] = data[i];
	}
	CFFT::Forward(pData, len);
	for( int i = 0; i < len; i++ )
	{
		pData[i] =  20*log10(abs(pData[i].re()));
```

```
                if( pData[i].re() > maxValues[i] )
                maxValues[i] = pData[i].re();
        }
        double x = 0;
        graph.moveTo(0, 600);
        for( int i = 1; i < len/2; i++ )
        {
                x = ((double)i / ((double)len / 2)) * 600;
                graph.lineTo(x, 500 - pData[i].re() );
                maxGraph.lineTo(x, 500 - maxValues[i] );
        }
        this->addPath(graph, QPen( Qt::black ), QBrush( Qt::NoBrush ) );
        this->addPath(maxGraph, QPen( Qt::red), QBrush( Qt::NoBrush ) );
}
```

Liite 8. Pääohjelman lähdekoodi.

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QGraphicsScene>
#include <usbHandler.h>
#include <oscScene.h>
#include <fftScene.h>
#include <QButtonGroup>
namespace Ui {
	class MainWindow;
}
class MainWindow : public QMainWindow
{
	Q_OBJECT

public:
	explicit MainWindow(QWidget *parent = 0);
	~MainWindow();

private Q_SLOTS:

	//called when connect button is pushed
	void onConnectButtonPush();
	//called when voltage/div dial is changed
	void onVoltageDialChange( int value );
	//called when time per div dial is changed
	void onTimeDialChange( int value );
	//kutsutaan kun dataa tarjolla laitteelta
	void onReadReady();
	//päivittää piirtoalueen
	void updateGraph();
	//hoitaa resoluutio menun klikkauksen
	void onResolutionMenuTrigger( QAction *sender );
	//hoitaa sample time menun klikkauksen
	void onSampleTimeMenuTrigger( QAction *sender );
	//hoitaa esijakaja menun klikkauksen
	void onPrescalerMenuTrigger( QAction *sender );
	//kutsutaan kun piirtoaluetta klikataan oikealla napilla
	//vaihtaa liipaisukohdan
	void onTrigPosChange( qreal y );
	//vaihtaa liipaisureunan
	void onTrigEdgeButton();
	//mittaus pisteen muutoksen hoitava funktio
	void onMeasPointChange( qreal v, qreal t );
	//vaimennuksen valitsimet
	void onAttenuation1x(bool toggled);
	void onAttenuation10x( bool toggled );
	//maatason kompensoitinappi
	void onGndButton();

private:
	Ui::MainWindow *ui;
	usbHandler *usb;
	QByteArray buffer;
	char charBuffer[8192];
	QTimer *timer;
	oscScene *osc;
	fftScene *fft;
	float volts[4096];
	int voltsPerDiv;
	int voltsPerPixel;
	float scrWidthTime;
```

```
        float samplesPerScr;
        float sampleDur;
        float sampleDiff;
        qreal trigLevel;
        double attenuation;
        int offset;
        bool trigEdge;
        float adcClock;
        int sampleTime;
        int resTime;
        int sampleDelay;
        int adcRes;
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtCore/QStringList>
#include <QtCore/QVariant>
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
        osc = new oscScene( this );
        fft = new fftScene( this );
        ui->setupUi(this);

        usb = new usbHandler( this );
        timer = new QTimer( this );
        ui->plotArea->setScene( osc );
        ui->fftNaytto->setScene( fft );

        trigLevel = 1000;
        attenuation = 2;
        offset = 1450;
        voltsPerDiv = 500;
        adcRes = 12;
        adcClock = 36000000;
        sampleTime = 3;
        sampleDelay = 5;
        trigEdge = true;

        scrWidthTime = ((float)20/(float)1000000) * (float)10;
        sampleDur = 1/adcClock * ( sampleTime + adcRes );
        ui->naytetaajuusLabel->setText( QString::number(1/sampleDur) );
        //naytölle mahtuvien näytteiden määrä
        samplesPerScr = scrWidthTime / sampleDur;
        //yhden pikselin leveys aikana
        sampleDiff = ui->plotArea->width() / samplesPerScr;

        //signaalien kiinnitys niitä hoitaviin funktioihin
        //aika per ruutu ruuvin pyöritys
        connect( ui->timeDial, SIGNAL( valueChanged(int) ), SLOT( onTimeDialChange(int) ) );

        //jännite per ruutu ruuvin pyöritys
        connect( ui->voltageDial, SIGNAL( valueChanged(int) ), SLOT( onVoltageDialChange(int) ) );

        //yhdistä nappi usb-handlerin yhdistä-funktioon
        connect( ui->connectButton, SIGNAL( clicked() ), SLOT( onConnectButtonPush()) );

        //usbin dataluku pääikkunan datanlukuun
        connect( usb, SIGNAL( readReady() ), this, SLOT( onReadReady() ) );

        //ajastin tahdistamaan päivitystä
        connect( timer, SIGNAL(timeout()), this, SLOT( updateGraph()) );

        //menu triggerit vastaaviin funktioihin
        // esijakajan menu
        connect(ui->menuEsijakaja,SIGNAL(triggered(QAction*)),this,SLOT(onPrescalerMenuTrigger(QAction*))
);
```

```
	//resoluution menu
	connect(ui->menuResoluutio,SIGNAL(triggered(QAction*)),this,
		SLOT(onResolutionMenuTrigger(QAction*)));

	//sample time menu
	connect(ui->menuSample_time,SIGNAL(triggered(QAction*)),this,
		SLOT(onSampleTimeMenuTrigger(QAction*)) );

	//liitetään liipaisukohdan päivitys siitä huolehtivaan funktioon
	connect(osc, SIGNAL(trigPosChanged(qreal)), this, SLOT(onTrigPosChange(qreal)));

	//liipaisureunanappi hoitavaan funktioon
	connect( ui->liipaisuReunaNappi, SIGNAL(clicked()), this, SLOT( onTrigEdgeButton()) );

	//mittauspisteen hoitava funktio
	connect( osc, SIGNAL(measPointChanged(qreal,qreal)), this, SLOT(onMeasPointChange(qreal,qreal)));

	//vaimennuksen valitsemisen hoitavat funktiot
	connect(ui->vaimennus10xRadio, SIGNAL(toggled(bool)), this, SLOT(onAttenuation10x(bool)));

	connect(ui->vaimennus1xRadio, SIGNAL(toggled(bool)), this, SLOT(onAttenuation1x(bool)));

	//maatason kompensointinappi
	connect(ui->maatasoKalibrointi, SIGNAL( clicked()), this, SLOT(onGndButton()));

	voltsPerPixel = voltsPerDiv / ( ui->plotArea->height() / 10 );
	if( voltsPerPixel < 1 )
		voltsPerPixel = 1;
}

MainWindow::~MainWindow()
{
	delete ui;
	delete osc;
	delete fft;
	delete usb;
	delete timer;
}

void MainWindow::onConnectButtonPush()
{
	usb->connectToDevice();
}

void MainWindow::onReadReady()
{
	usb->getData(charBuffer);
	buffer = QByteArray(charBuffer, sizeof(charBuffer) );
	//kutsutaan graafin päivittävää funktiota ajastimen kautta.
	timer->setSingleShot(true);
	timer->start( 1 );
}
void MainWindow::onTimeDialChange( int value )
{
	ui->timePerDivEdit->setText(QString::number( value ) + " µs" );
	//naytön leveys aikana
	scrWidthTime = ((float)value/(float)1000000) * (float)10;
	//naytölle mahtuvien näytteiden määrä
	samplesPerScr = scrWidthTime / sampleDur;
	//yhden pikselin leveys aikana
	sampleDiff = ui->plotArea->width() / samplesPerScr;
}

void MainWindow::onVoltageDialChange( int value )
{
	ui->voltagePerDivEdit->setText( QString::number( value ) + " mV");
	voltsPerDiv = value;
	voltsPerPixel = voltsPerDiv / ( ui->plotArea->height() / 10 );
	if( voltsPerPixel < 1 )
```

```
                voltsPerPixel = 1;
}

void MainWindow::onResolutionMenuTrigger(QAction *sender)
{
        char command[2];
        command[0] = 'W';
        command[1] = (char)(sender->text().toInt());
        usb->sendCommand( command );
        adcRes = sender->text().toInt();
        //muunnosaika on kellojaksoina on näytteistysaika + resoluutio
        sampleDur = 1/adcClock * ( sampleTime + adcRes );
        ui->naytetaajuusLabel->setText( QString::number(1/sampleDur) );
        //naytölle mahtuvien näytteiden määrä
        samplesPerScr = scrWidthTime / sampleDur;
        //yhden pikselin leveys aikana
        sampleDiff = ui->plotArea->width() / samplesPerScr;
}

void MainWindow::onPrescalerMenuTrigger(QAction *sender)
{
        char command[2];
        command[0] = 'D';
        command[1] = (char)(sender->text().toInt());
        usb->sendCommand( command );
        adcClock =  84000000 / sender->text().toInt();
        //muunnosaika on kellojaksoina on näytteistysaika + resoluutio
        sampleDur = 1/adcClock * ( sampleTime + adcRes );
        ui->naytetaajuusLabel->setText( QString::number(1/sampleDur) );
        //naytölle mahtuvien näytteiden määrä
        samplesPerScr = scrWidthTime / sampleDur;
        //yhden pikselin leveys aikana
        sampleDiff = ui->plotArea->width() / samplesPerScr;
}

void MainWindow::onSampleTimeMenuTrigger(QAction *sender)
{
        char command[2];
        command[0] = 'T';
        command[1] = (char)(sender->text().toInt());
        usb->sendCommand( command );
        sampleTime = sender->text().toInt();
        //muunnosaika on kellojaksoina on näytteistysaika + resoluutio
        sampleDur = 1/adcClock * ( sampleTime + adcRes );
        ui->naytetaajuusLabel->setText( QString::number(1/sampleDur) );
        //naytölle mahtuvien näytteiden määrä
        samplesPerScr = scrWidthTime / sampleDur;
        //yhden pikselin leveys aikana
        sampleDiff = ui->plotArea->width() / samplesPerScr;
}

void MainWindow::updateGraph()
{
        int i = 0;
        //laitteelta tulevat arvot ovat 16 bittisiä mutta ne tulevat 8 bitin pätkissä
        //käydään puskuri läpi
        for( int a = 0; a < buffer.size(); a += 2 )
        {
                //muunnostulos on kahdessa peräkkäisessä tavussa
                //joista toinen on siirretään 8 bittiä vasemmalle
                volts[i] = (float)(((buffer[a] * 256) + buffer[a+1] ) - offset ) * attenuation;
                i++;
        }
        //etsitään liipaisukohta
        //käydään arvoja läpi kunnes tullaan liipaisukohtaan
        //hylätään muutama tulos alusta, ovat jostain syystä hieman epävakaita
        i = 50;
        bool liipKohtLoyt = false;
        if( trigEdge )
        {
```

```
                bool kasvava = false;
                while( !liipKohtLoyt )
                {
                        if( (volts[i+1] - volts[i]) > 0 )
                                kasvava = true;
                        else
                                kasvava = false;
                        if( kasvava )
                                if( volts[i] > trigLevel )
                                        liipKohtLoyt = true;
                        i++;
                        if( i > buffer.size()/2)
                                break;
                }
        }
        else
        {
                bool laskeva = false;
                while( !liipKohtLoyt )
                {
                        if( (volts[i + 1] - volts[i]) < 0 )
                                laskeva = true;
                        else
                                laskeva = false;
                        if( laskeva )
                                if( volts[i] < trigLevel )
                                        liipKohtLoyt = true;
                i++;
                if( i > buffer.size()/2 )
                        break;
                }
        }

        if( ui->skooppiTab->isVisible() )
                osc->myUpdate(volts, 4096, i, voltsPerPixel, sampleDiff );
        if( !(ui->fftTab->isVisible()) )
                fft->myClear();
        if( ui->fftTab->isVisible()  )
        {
                fft->myUpdate( volts, 4096 );
                ui->fftNaytto->update();
        }
        //tyhjennetään puskuri jottei piirrellä samoja toiseen kertaan
        buffer.clear();
        ui->plotArea->update();
        usb->sendCommand("b");
}

void MainWindow::onTrigPosChange( qreal y )
{
        y = ui->plotArea->height() - y;
        trigLevel = y * voltsPerPixel;
}

void MainWindow::onTrigEdgeButton()
{
        if( trigEdge )
        {
                trigEdge = false;
                ui->liipaisuReunaNappi->setText( "Laskevalla reunalla" );
        }
        else
        {
                trigEdge = true;
                ui->liipaisuReunaNappi->setText( "Nousevalla reunalla" );
        }
}

void MainWindow::onMeasPointChange( qreal v, qreal t )
{
```

```
	ui->deltaVlabel->setText( "dV: "+QString::number(v * voltsPerPixel ) + " mV" );
	qreal pikselinLeveysAika = scrWidthTime / ui->plotArea->width();
	QString temp = QString::number(pikselinLeveysAika * t, 'e', 3 );
	ui->deltaTlabel->setText( "dT: " + temp + " s" );
	ui->taajuusLabel->setText( "f: " +
	QString::number( 1/(pikselinLeveysAika * t), 'e', 3 ) + " Hz" );
}

void MainWindow::onAttenuation10x(bool toggled)
{
	if(toggled)
	{
		attenuation = 10;
		ui->vaimennus1xRadio->setChecked(false);
	}
}

void MainWindow::onAttenuation1x(bool toggled)
{
	if(toggled)
	{
		attenuation = 2;
		ui->vaimennus10xRadio->setChecked(false);
	}
}

void MainWindow::onGndButton()
{
	offset = volts[100];
}
```